active computers

Der BASIC-Interpreter im Laser 110, 210, 310 und VZ 200

Gerhard Wolf

Mein Home-Computer

Gerhard Wolf

The BASIC interpreter in the laser 110, 210, 310 and VZ 200

HC- My Horne Computer

Gerhard Wolf

The BASIC interpreter in the laser 110, 210, 310 and VZ 200

Structure and operation

VOGEL-BUCHVERLAG
WÜRZBURG

From the same author are published:
ROM Listings for Laser 110, 210, 310 and VZ 200 (HC - My Horne Computer)
ISBN 3-8023 0852-2

The Laser-DOS for Laser 110, 210, 310 and VZ 200 (HC - My Horne Computer)
ISBN 3-8023-0868-9

CIP short title recording of the Deutsche Bibliothek **Wolf,**

Gerhard:

The BASIC interpreter in the laser 110, 210, 310 and VZ 200: Structure and Operation/ Gerhard Wolf. - 1. Aufl. - Würzburg: Bird, 1985
(HC - My Horne Computer)
ISBN 3-8023-0874-3

ISBN 3-8023-0874-3
1. Edition. 1985

Printed in Germany

Envelope design: Bernd Schröder, Böhl
Manufacturing: Alois Erdl KG, Trostberg

The small computers of the series LASER 11@, 21%, 310 and the V2200 owe their popularity not least to the comfortable and extensive BASIC Interpreter, which is located in memory modules (ROl'ls) i111 interior of the computer and is fully available to all users in all its functions after switching on.

It is here LIII a slightly modified version of the well-known and worldwide MICROSOFT BASIC, which is part of a basic operating software. This operating software consists of a basic operating system and the extensive BASIC interpreter.

A floppy operating system (DOS = Disk Operating System) is used as a backup. This is also stored in ROtt blocks, but is not housed in the main computer, but is located in the floppy control. By connecting the floppy control to the system bus, these ROtt modules are activated and add the necessary Soft.are· Operation of the floppy drives to the base system.

A further performance increase can be achieved by an EXTENDED RASIC package developed in Germany, which extends the language range of the built-in BASIC interpreter by more than 3 commands, among them such powerful ·PLOT, PAINT, CIRCLE, RENUl'IBER and many more.

The aim of this book is to describe the essential functions of the BASIC-ROl'l so that you can better understand the internal processes in your computer and make the most of all functions. The book is also intended to open up the possibilities for dell AsSl!llbler/Kaschinenprograma experts to use functions of the BASIC-ROl'l in their own progrees, be it to perform simple data conversions, to use the input/output interfaces or atheatic functions (e.g. sinus, cosine USIII,J not to eat prograMieren yourself.

Not every routine can be described in detail. However, the description contained herein should be sufficient to allow for further detailed investigations in the ROtt collection itself.

A detailed documented ROii collection and a detailed description of the floppy operating system!IIS for LASER 118-318 and VZ28111 appeared as independent volumes in Vogel publishing.

No thanks to Mr. Dieter Effkl!llann for his support and cooperation to chapter 12 'Functional derivations' and 111a son Rainer for testing the programme examples and for extensive proofreading.

1. las is a working gyte

A computer without any kind of operating system is a useless box filled with electronic components.

The need for an operating system arises solely from the need to establish a communication capability between the computer and its environment. This includes continuous monitoring of all inputs, e.g. the keyboard, in order to receive instructions and data, and also the output of data, e.g. on a connected screen, to display or transmit results.

If you want to type in a BASIC programme to your LASER computer and run it afterwards, there is already a **PrograH** on your computer, which will first take your input and get it to the right place in the memory. The running of the programme also requires strong support from this internal programme. One such programme in the computer is the operating system,

There are now thousands of different operating systems, from the simple system in RO# of Hone and Personal computers to complex structures that can occupy entire disk units in large-scale computing.

The differences are due to the different computers 11it of the most different hardware equipment, but also to the requirements that are placed on such a system. The operating system of a computer for process control looks very different to that of a commercial application, although both may run on the same hardware.

For this reason, it is not possible to establish a precise definition of an operating system.

Despite all the differences, however, the basic building blocks of the operating system- III! similarly and in general, can the following components be detected functionally in the majority of systems?

1. A nonitor programme that constantly monitors all system inputs (e.g. the keyboard query).

2. Device driver routines that meet the specific physical needs of connected peripheral devices such as keyboard, video, cassette, floppy or printer.

3. General service routines that initialise system functions after entering certain commands (in LASER e.g. LIST, CLOAD).

4. Language **translators (compilers**, assemblers, interpreters) that enable the use of a Programme111 language (e.g. BASIC, PASCAL, FORTRAN).

5. Runtime support routines for the programming language.

   These are among others the mathematical and mathematical functions for which finished routines ir, operating system are available, which are only called by **the** respective language.

6. Device and Storage Management helper routines;

   These maintain internal tables, manage the different storage areas and control access to the I/O devices.

The following sections identify and describe these components within the LASER operating system.

Since LASER computers have the operating software in ROM modules within the computer, it is immediately available in all of its functions after power-up.

On other computers and operating systemsNo the operating software first read in from an external memory (disk, tape, floppy, cassette) who the, **but** also a small programme is required. This is referred to as the Urloder <IPL = Initial Progra Loader), which exists somewhere in the system or is to be entered by hand.

The operating system of the LASER computers 11@-310 and the VZ208 is a self-contained programme package that can be run independently (stand alone system),

It includes a BASIC interpreter as a language translator, as well as the necessary support and assistance routines to run BASIC programmes. Furthermore, it offers the possibility to save prograne on cassette and to reload from cassette.

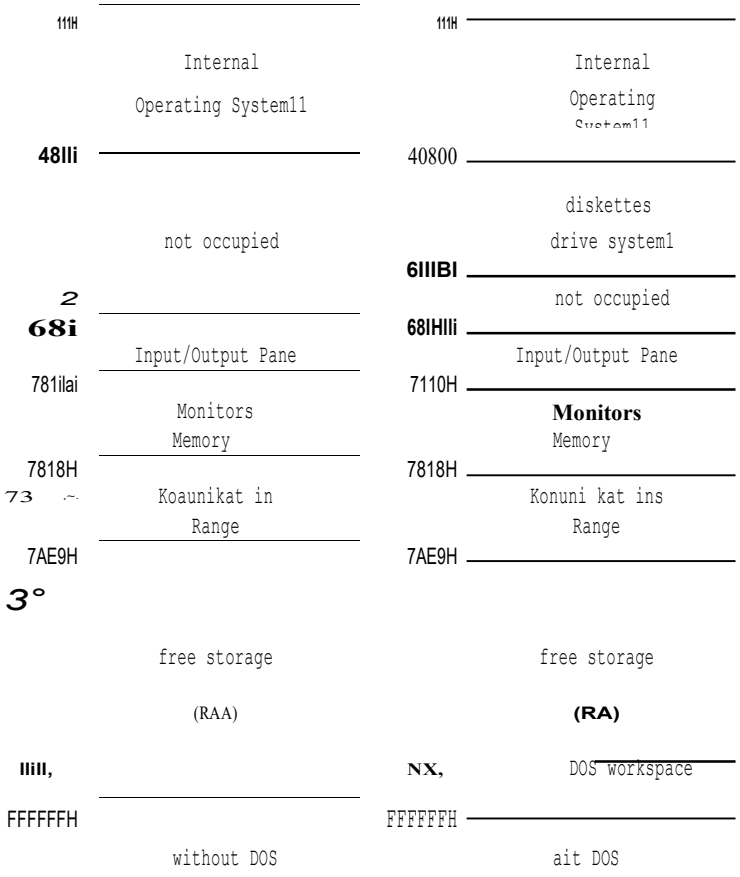
The floppy operating system is only available with the floppy drive connected. It extends the language scope of the BASIC interpreter and floppy disk editing applications and offers the ability to save and load programmes from floppy disks.

If a floppy drive is attached, the floppy operating system will be connected when the computer 11it dell internal operating system is turned on.

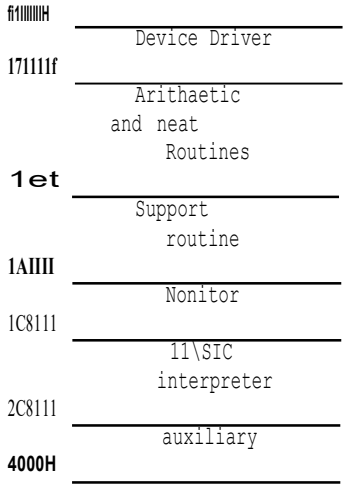
The floppy operating system has its own interpreter to detect and edit the additional commands, but makes intensive use of support and helper routines of the internal system and is therefore not a self-running programmeMAppliance,

EXTENDED BASIC is a language extension of the BASIC interpreter and offers the convenience of additional commands, especially in graphics and the Progr on development. It is available on a cassette or floppy depending on your computer's configuration level and must be loaded into the computer in addition to the internal operating system.

- [


111H
Internal
Operating System11
48lli
not occupied
2
68i
Input/Output Pane
781ilai
Monitors
Memory
7818H
73
Koaunikat in
Range
7AE9H
3°
free storage
(RAA)
lliill,
FFFFFFH
without DOS
111H
Internal
Operating
System11
40800
diskettes
drive system1
6IIIBI
not occupied
68IHIli
Input/Output Pane
7110H
Monitors
Memory
7818H
Konuni kat ins
Range
7AE9H
free storage
(RA)
NX,
DOS workspace
FFFFFFH
ait DOS

The first 1k are assigned exclusively internal operating systems. If you look at the basic building blocks of an operating system, the situation in this area is similar to the one below.
However, no clear separation was performed, single code elements; which are assigned a fixed element by their function, were wildly scattered in memory. In front of all at the end of the RONBerach you will find a series of backpacks', which were created when later versions changed.



If a floppy system is connected, the following B8K are populated by the floppy disk operating system, otherwise the address range up to 688MH is not populated with memory modules.

Below is a section1 that is titled 1it 'Input/Output Range'. There is no memory hidden behind this address. If the input/output blocks are appropriately addressed, many direct interfaces are addressed. So is z.. wired the keyboard as a matrix in this address space, where each individual key is directly interrogable.

The following is the video memory ait 21(, behind it is a 2K RAMBaustein which is constantly read out and displayed by the image generator.

In the following communication area, BASI interpreters create and use work areas, address pointers and management tables.

Behind the Klllllllllt.mikationsbereich is the free storage. This is the area where BASIC-Progra1111e and its variable are stored. There you can also load your Asserbler or l'laschinenpr ogr anme and bring it to a finish.

For the attached floppy drive, a 318-byte working area is created at the end of the DOS memory during system initialisation, which for the DOS is similar to the communication area for the BASIC. If no floppy drive is attached, the free memory will reach the end.

4. The On-[Output Pane (6800H- 6FFFH)

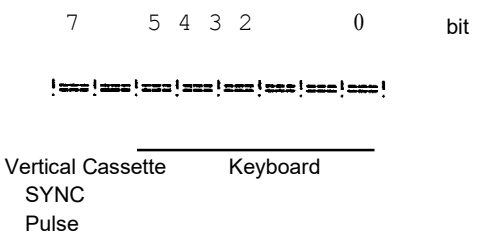
This memory area is used to directly control input/output blocks such as keyboard, speakers, cassette and image generator.

If input or output addressing is used, the respective bits have different meanings.

These addresses are used to read the keyboard matrix, cassette input and the vertical SYNC pulse of the image generator.

On the output side, the image generator, the speaker and the cassette are controlled and addressed via this address area. Since an output value cannot be read back 11, the operating system always keeps an up-to-date copy of the output value in the address 783BH of the Know information area and uses it as a reference if another output is to be made.

Except for the keystroke it is irrelevant which address of the Adrelral.lllf!S you address, only the top 5 adrelbits are evaluated.



Output

Keyboard,All

The keyboard forms in dell Adreilralll b800H to bBFFH a tatrix of 8 rows and columns, organised as follows:

A8 - A15 = 68

								D8	Dl	D2	DJ	D4	D5	
A7	A	A5	A	A3	A2	AI	A							
0	1	I	I	1	1	1	1	H	L		K		J	= 687FH
	0	1	1		1		1	V	0	Ret	I	p	U	= 68BFH
	1	0	I		1		1	6	9		8	O	7	= 68DFH
	1	I	1		1		1	N				Spe	M	= bBEFH
			0	1	1	1		5	2		3	1	4	= 68F7H
			1	8	1	1		B	X	Shtt	C	or	V	= 68FBH
			I	I	8	I		6	S	Ctrl	D	A	F	= 68FDH
						0		T	w		E	Q	R	= 68FEH

You see, that a "@ in an address bit contains the line to be read, Is one of the read data bits = 1, as is the Associated Key in Pressed the intersection of the

If you set the *more than one address bit=*, then *several* lines will be read at the same time, so you can read out *the* **6880** address hit at once.

Hasset tefjppebe

bad" Bit *b* of address 688111H (or any other of range 6888H-6FFFH) are read the Pulse **voa** cassette recorder.

3ildscirsetsueruns

bad" the adrefbel"eich 688111H-bFFFH can be transferred control functions to the image generator.

Bit J = Slideshow 1 = Text
1 = Graphics

Bit 4 = Display Colour s green
1 = red

Volume Output

With the bits I and 5 you control the built-in small speaker., The two bits **must** be **internally** complementary; otherwise you will never produce a sound (i.e. bit 0=f and bit 5= or vice versa),
By switching these bits in a besti111111th frequency the pitch is **fixed.**

cassette output

via the Bi tpasi tions 1 and 2 of the AdrelraulleS 681111H-6FFFH, you can output data bits to the cassette, 1110for both bits should always be the same.

For all three outputs (image generator, loudspeaker, cassette>, 1you must always observe the previous status and change only the bits you want to change. Therefore, the RON routines in address 783BH of the costing area **are** always kept by a current copy of the output byte. You should also note and understand this if you want to spend something directly on these building blocks.

Vertical SH:.-/lse

The Yertikal SYNC pulse generated by the image generator and others (all 2 ѕ in PAL) **is** usually used to generate an interrupt in the CPU. As described in detail later, this interrupt is used to synchronise the via carry in the Video-RA, including getting a flicker-free image.

This pulse is also palpable over bit 7 of the 681111H-6FFFH ear range, even if the interrupts are disabled. You can use it to synchronise your machine programmes even when the interrupts are switched off, without having to perform your own · interrupt treatment,

3. he Mil±shirsreicher

Ia Screen Savers are the characters to be displayed on the screen. This memory area is constantly sampled voii image generator and the picture information t:1 is displayed on dea Monita. The screen saver occupies the 7888H+-77FFH (= 2K).

I1 Text110dus only uses the first 512 bytes, each byte being able to contain a character to be displayed. This corresponds to an output capacity of 16 lines a' 32 characters.

Note that the$ uses its own screen codes that do not correspond to **the** ASCII code here.

Two display colours can be selected via the screen control, these are green and red. There are again1.111 two display modes, whereby the colour selected as background colour or as font colour.

The display type is controlled via the address 7818H of the Komuni Action range. Between the modes of presentation can also be used via the inverse Toggle character representation.

Within Text Output is also a block graphic in 8 different Far ben alike

Character Table:

II	=	a	8B	=	K	1	=	V	21	=		2C	=		37	=	7
0I	=	A	€	=	L	17	=	W	22	=		2D	=		38	=	8
02	=	B	D	=	"	18	=	X	23	=	1	2E	=		39	=	9
83	=	C	IE	=	N	19	=	y	24	=	$	2F	=	I	3%	=	
%4	=	D	IF	=	0	1A	=	Or	25	=	1	30	=	0	3B	=	i
05	•	E	11	=	p	1B	=	[	2	=	&	J1	=	1	3C	=	<
0%	=	F	11	=	Q	1	=	\	27	=		32	=	2	3D	=	=
e7	=	6	12	=	R	1D	=	]	28	=		13	=	3	YES	=	>
08	=	H	lJ	=	S	1E	=	A	29	=		34	=	4	3F	=	?
99	=	I	14	=	T	1F	=		2A	=	f	35	=	5			
84	=	J	15	=	U	28	=		2B	=	+	36	=	6			

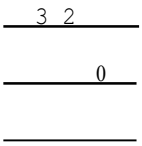
The codes 48H to 7FH represent the same inverse characters.

A block graphic character is marked by Bit7 =1. The Bi positions 4, 5 and determine one of eight colours

bit	6	5	4		
	0	**0**	**0**	=	green
	0	**0**	1	=	yellow
	0	1	**0**	=	blue
	0	1	1	=	red
		0	**0**	=	**beige**
		0	1	=	light
			0	=	pink
				=	orange

In bit positions - 3 **is** the form of block graphics **besti111t.**

Each of the four bit positions is assigned to a quarter of a character position. The following Fig.mg shows the mapping ***of the*** individual bits:



Graphic-#mode uses the entire ZK of the screen memory to achieve a resolution of 128 x 64 points (pixels) in IIII.

The information of four pixels is stored in each byte, with each pixel assigned 2 bits.

It is possible to display four different colours to these 2 bits, which can be chosen between two colour sets via the screen control (display colour) IIII!!'den.

68111111H		Bit **4 = 1**	Bit 4 = 1	
	III	green	**white**	(Background)
	01	yellow	turquoise	
	11	blue	orange	
	11	red	violet	

6. The launi tat area

The Command!larea is the l'erkzettel of the operating system and is located in the range 7811111H to 7AE9H,

It contains tables, pointers, and addresses that are created and managed by the operating system.

There are also work areas and cache areas required to perform arithmetic and input/output operations.

Within the operating system there are a whole series of RAE expansion outputs," which are CALL calls to a 3-byte range i111. Nonetheless, the outputs i111 Kma.inikationsbereich are occupied with a RETurn,

The RA} expansion outputs allow you to connect your own port routines to the various operating system functions. EXTENIJED BASIC uses this very intensively, the DOS also connects to the BASIC interpreter via such an extension output.

Some help routines that jNeils modified before de ¨ call must be also accommodated in the communication area. They are brought there after the computer is switched on during the system initialisation from the RON, one of these routines is a subprogramme · of the division function. This is modified from the ROii routine to the division and called to perform special subtraction and comparisons.

For a description of each field in the communication area, see Chapter 9 "Addresses and Tables of the BASIC Interpreter · ,

The free storage device is available for you to load and run your own BASIC and **machine programmes**.

Match **the** activation of the computer, the free memory extends from the end of the configuration area to the end of physical memory, or until the start of the DOS workspace at activated. Floppy Drive.

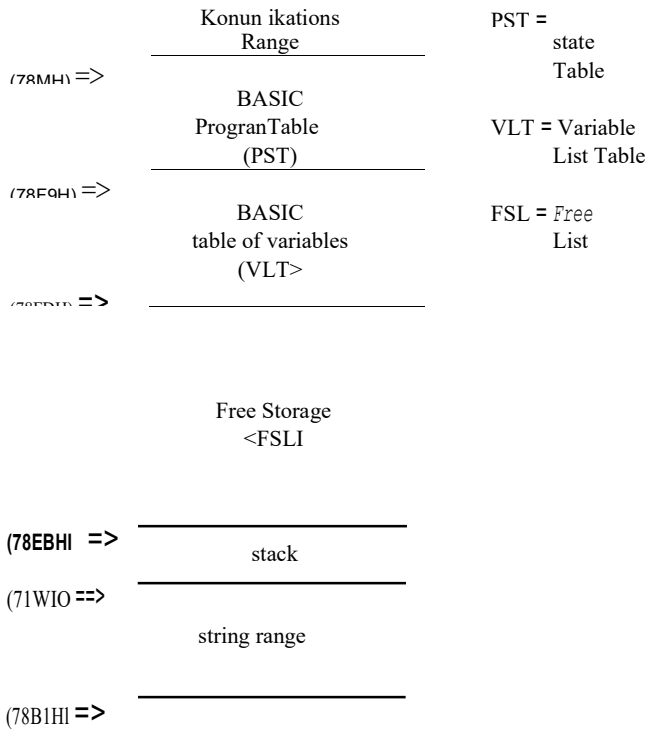
The limits of the free memory are **displayed** in two adres of the communication area:

78A4H/78A5H contains the initial address
78B1H/78B2H containing the final address

By manipulating these pointers, you can reduce the free memory from the top or from the bottom. This is useful if you want to protect a **busy** memory area from **the** BASIC interpreter (see 00SAworking area),

If you use the free storage unit exclusively for machine progranes, then **the entire room** is fully available to you, the machine programme is responsible for the management of this space itself.

For BASIC prograses, the BASIC interpreter of the RON outperforms the memory**#anagenent** door to the free storage area. It **is** divided into different sections and structures and dynamically aged ver111.



Note that the *areas* shown above are not fixed addresses to9eordered. These areas are dynamic, i.e. their grids and *borders* are fluid and are constantly adapted to current needs.

Look at the Progrua table. This contains the loaded BASIC **programme.** If you insert a programme line there, so the Prograa table 11.1s stretches, at the same time *the* beginning *of the* variable table moves, which connects directly to the Prograa table, define a new variable in your Pogra, so the variable table enlarges.

Because the boundaries *of the* single111111 areas are divisive1 .their current addresses **are** listed in adre$zeigern of the caching area. This allows moving dl!r tables according to the requirements **of the progras** and at the same time offers users the legibility *of* ltlnipulation.

The Prograaa table (PSTl) contains the individual rows of the BASIC profile in a coordinated format. Copied means that within the rows, the BASIC keywords are replaced by 1-byte hexadecilates (TKEN)

The start address of the Prograna table is in Adre$zeiger 7844/ of the canunication area. Make the loading of a programme remains constant. If the beginning of the free memory is not unified IEXTENDED BASIC does something like this for example)+ you will find there the entry 7AE9H.

The final address of the programme table varies with the Gr@e of the propane, it is identical to the initial address of the variable table and is found in the address pointer at 7F9.

The variable table (VI..T) contains noses and values of all variables defined in a BASICProgr of variables. It is divided into two sections, Section 1 contains the simple variables and Section 2 diaensioned variable, i.e., sodium.

Variable names and values are entered in the order in which they occur at the expiration of a propane. Each new variable will enlarge the table. Once defined variables remain in the table until the system reinitializes, i.e. you cannot delete them, but at most change their value.

From the end of the variable table to the beginning of the stack area, there is the remaining free memory, which originally stretched from the end of the Kc:aunication range to the end of the memory. You can see from the picture that this free-space from the top {lower adrerau) is narrowed by the PrograTable and the Variables table and from the bottom (top adrspace by the String area and the Stack area. This works fine until the end of the variable table collides with the stack area. In such a case, the error message 'OUT OF l'ENORY' is reported.

The stack area is very dynamic. It constantly serves the operating system as a cache for return addresses and register contents. Each CALL, PUSH, or RST command increases the stack size by 1,111 bytes, and each RET or POP command decreases the stack size accordingly.

From BASI interpreters, entire memory blocks are written on the stack. Each FOR/l'EXT loop puts a 1B-byte block on the stack, and each 60SUB command puts a 7-byte block on it.

The last section shown behind a stack area is the StringRange. With the exception of the text variables defined in the programme (included in quotation marks in the programme text), all text variables (=strings) are included in the programme. Range filed. The variable table contains the reference address for the string range or the programme table, where the corresponding text can be found, for the text variable as a value.

The length of the string range i.is defined at initialisation with 50 bytes. However, Nit dea CLEAR-K01111ando may vary.

8. Operating System Features

In the previous sections, there was talk of the basic building blocks of an operating system. In addition to this breakdown, the LASER operating system may be divided into the following seven functional divisions:

1. System Initialisation 2,
Input Routine
3. Interpretation and Execution Control
4. Command and code execution
5. Arithlletic and Atheatic Routines
Input/Output - Drivers
7. **System** Features

Each of these functional areas should now be examined in greater detail.

8strInitialisation

The first section of this book mentions that the LASER operating system is in RON modules and is available immediately after power-on. This is only half the truth, Although this system does not require any additional components to be reloaded from any0her, however, some initialisation functions are to be performed, such as setting up the Kounikat ion area before the operating system it can work with.

What will happen with the system initialisation?

After the computer starts its i1111er programme execution at the absolute address I of the RON area, where the image generator is switched to the text bus and branched to address 74}.

At 674H the RON content is transferred from address 6D2H to address 787H to the 111nikationsbel"eich from 7811111H to 7835H which initializes the addresses for the Restart vectors 8, 1, 18 and 20, so you can jump into the current Rffl routines. The Restart vectors 28, 31 and J8 occupy the with RETurn errors (no function),

It will also fill the 'devices control blocks' (Device Control Blocks) for keyboard, monitor and printer **ait** outputS111erten (7815H - 7B2CHl, The memory range 7836H to 7B5CH ・Will be deleted (Nlil,

For **further** initialisation, branch to address 75H, where whoever performs the **following** functions:

The RON range from 1BF7H to 191DH is transferred to the configuration range from 7881H to 7BA6H. In this area you can find the help routine for the division. It also initializes some switches and pointers, including the start address of the Pr gra table at 78AAH.

The input/output buffer from address 79E8 is initialised, its initial address is entered in buffer pointer 7BA7H and a header of JAH-011i-2CH is written before the buffer. This buffer is used, among other things, to cache each programme line that is input and output.

Address vectors for special floppy commands from address 79524 to 7945} are pre-populated **with a** jump command to address 1D2H (JP lll1D2Hl). This causes the error message "DISK COH#AND ERROR if one of these vectors is triggered.
These disk vectors are a relic of a previous operating system operating system input computer and are not used by LASER-Conputer.

The RAH extension outputs 7A} to 79} are pre-populated **with** RETurn (C9).

Address 7AEBH (directly in front of dl!IR free memory l ・is loaded ・it 8111H and the stack pointer **is** first initialised 1it 79F8H (in the input/**output buffer)**. This happens because the initialisation with CALL calls is now continued and a stack to save the return address is required.

A subroutine at lBFH (within **the NE** configuration) is called. There, the stack layer is unset to 7M45H, **an address** area ・ **Free** Storage Area.

The string cache {from address 78B5}) **is** marked as empty, **because the** adre$pointer **is** set to the first entry at 7BBJH.

The output device is set to the screen and, if necessary, a carriage return is written to a connected printer.

The indexing lock is reset and the end of the stack area ait 8 is marked.

The video memory is cleared (CS).

The memory end address is obtained and stored in 78B1H. For the string area, 5 bytes a are reserved for the end of the free-memory and the start address of the string range is entered in 784H.

The subroutine 1B4DH ltEW) is called.
There · The TRACE-Funktian and the AUT<>-Nodus are switched off,
The beginning of the Pragraa table is initialised with ilJtl-Ml, UII to mark them as empty.
The address pointer to the variable table (=end address of the **progrel table) at** 78F% is set to the start address of the **progreeter table** + 2, Since the variable table is also empty, the end addresses of both parts are set to the same value at 7BFJH and 78FDH.
The TypeCode table from 7981H · is set to 'simple accuracy' for all variables.
The stack area is set up in front of the string area.

Ia On this is called a subroutine at 3484H, There is checked (CALL WFMH), if at the same time the CTRL key was pressed with the switch on (green background) and accordingly the background flags 7818H and 7819H set.
To create the correct background, the screen will be deleted.

via the input/output addresses (681111) · The default 28H is set and also marked in memory at 7SJBH, This means 'display colour = green' and 'text mode',

The basic values for the delay counter (7344) and the flash counter (7841Hl) are set.

The colour value 111is set to 'yellow' (7846H) and the interrupt vector in 787DH is set to RETurn (C9),

Now finally the output of the introductory text takes place "VIDEO TECHNOLOGY .... • and the Z89 is switched to InterruptMode 1. This means that the programme address 38 will be jumped in case of a cascading interrupt, such an interrupt will occur if interrupts are allowed (Eil, every **28** milliseconds.

From address 068EH the last phase of the initialisation takes place. There one after the other, it is checked whether a ROl'l cartridge slot is present in the address ranges 4NIH1 **6l0illi or** 801110H (e.g. the DOS from address 48H), such cartridge slot **must be placed at** the above. Begin addresses in the sequence AA}-55HE7H-18H.
If this sequence is determined at one of the addresses, the following address is referenced, otherwise the 11it **of the** BASIC input routine will continue.

In this way, it **is** determined, among other things, whether a floppy system E!III is connected. The floppy add-on routines are housed in **ROl'ls** in the floppy disk control. Addressing starts **at address 4NIH, where the** first four locations are AM-5HE7H-IH. If the floppy disk system is connected, this byte sequence is found at 48ml and the **Pgr ana** for initialising the DOS from address 4884/ continues.

The Input OutTime

The input routine is similar across all operating systems. Their function is to overwrite keystrokes and respond to received commands,

In the LASER 118, 210, 3l@ operating system and the VZ280 both SystemsKoandos and BASIC programme lines are processed by the input routine.

The input routine is introduced at address 1419, also known as the beginning of the BASIC main loop or BASIC-ilarastart address.

The message READY" **is** displayed there and jumped over the **RAN** extension output at 78ACH !initialised ■ it RETurn>.

The functions of the input routine can be divided as follows:

1, Read line from keyboard
2. Replace BASIC keywords in line with TOKEN 3, check if a direct command is entered
   (BASIC line without line number).
   If so, continue with ,
4. Move row to programme table.
5. Back to 1.
. Interpreting and Executing

The input loop starts at *the* address 1A33H. If the system is i111 AUTOModus, the line contents will be output first and if this is already available via the programme• also the line contents.

With a CALL to the routine at 03E3H 111a line is taken from the keyboard and transferred to the input/output buffer from address 79E8H. If the 1111t line input of the BREAK key has been completed, the line input will immediately return to the beginning of the loop (1A33H), i.e.

A line number entered will be converted from ASCII to binary <CALL 1E5AHJ.

In the subroutine ab **IBCIIH** the input line **is** examined and all IASIC keys1110rte contained in it are replaced by 1-byte hexadecimal connoisseurs, the 'TOKENs'.

Then a jump to RAN extension output at 79%2H, a good opportunity, 1111mn iaan itself still wants to handle the input line (connection of a machine routine). This includes, among others, EXTENDED BASIC for detecting and setting **the** additional commands.

1AA4H **will** check if a line1111ter **has been** entered. If not, it is **now** a direct:oaando, In this case, the interpretation and execution control (1D5AHI) is branched immediately.

If there is a LineUMer, the row you entered will be added or inserted at the correct location of the Profiles table.

If **a** line of the same **mumer already exists**, then this **will** be deleted previously **with the** routine at 21!E4H.

The end address of the programme table (7F9) is increased and the length of the entered line is increased l1AC2H ・・・・・・J and 1it of the routine at 1955H Space is created for the new line, inde!t the lines behind it 11it higher linesUNer are moved up into memory.

Starting with 1ADIH, the preparation and transfer of the line from dell I/O buffer to the programme table begins.

If the line you entered is empty, this will be noticed at 1ARFH and the transfer routine will be skipped.

The **reference addresses** for line chaining are renewed in the entire **PrograM** from the routine at 1AFCH. 1B5DH in the NEW routine 111Zll Abschlu8 is still called, **and** to delete **the** variable table and reset some flags and connoisseurs, so that no FOl"execution **of the Programs nit** CONT **is possible** after inserting or changing a line,

Then a return to the beginning of the loop **to** insert the next line.

ed

The AUTO command is only available in EXTENDED BASIC, but can be **turned on in normal BASIC with the help of a POKE 30945.1**, if is satisfied **with the** default values (starting value = **1, step size** = 1).

I・ AUT<Hlodu1 is provided the line number to be issued in address 78E2H/78E3H and the increment value at 78E4},
The lines **i** AUTO-#Mus are read in section lAJFH to 1A76H.
'**This** output **is** the line number from 7BE2H/78E3H and, if this line is **already** present (search **with** 1R2CH), with 2E53H also the line content.

Reading is also done via BJEJH. If the BREAK key is pressed, the AUT0-11odus is also turned off in addition to ignoring any input. Make **the** row read (from 1A6611l the time stored at 78E2H/78E3H **will be** read and the value from 78E4H increased.

The further treatment of the row is done, as with the input without AUTO, starting from 1A81H.

Anger kungk
For" LASER 118, 218, 31 and VZ200, a number of BASIC keywords are not encoded, although the required execution routines and TOKEN are available (see command table starting at 1650H), probably for copyright reasons.
Hit EXTENDED BASIC will open some of these commands and functions, among many additional ones, and make available ge111eight.

Interpretation and Execution Control

System111k011111andos and BASIC commands are executed through interpretation.

Interpretation means that all Kouandos and all BASIC lines are analysed only at the execution time of the operating system and the required operations are initiated,

Such a procedure is common at the system command level. A Kon ando is read in, interpreted and the corresponding execution routine kicked in,

To execute a programme with the help of interpretation is generally limited to personal and home computers and there only for a few languages, such as BASIC or LOGO,

The alternative to interpretation is the previous K011piling by means of a COllpil er.

Copilers translate the source code, which are the input lines in the corresponding language (FORTRAN, COBOL, PASCAL, Pl.1, etc.), in directly executable machine code, called object code.

Such an object code is loaded and launched into memory at run time with the help of a loader (part of the operating system). Match the start runs such a **programme** almost completely independent of operating system.

The LASER calculators 111, 211, 311 and *the* VZ208 work in BASIC only **with the** 11method of interpretation. If you want to use directly executable machine code, you have to create it with an asseabler or enter it directly.

The main tool of an interpreter is first the comparison. A typed programme line of a BASIC programme is checked character by character and searched for BASIC keywords such as IF, THEN, FOR, NEXT, GOTO, etc. Each keyword found is replaced by a unique hexadecimal digit called TOKEN (e.g. CLS = AM, IF = 8FH, GOSJ = 91H, CLOAD = B9Hl. After this, the Progra table will contain the Progra row, which is handled in this way.

This function takes place at the time of the prograNo and relieves the execution control of this time-consuming work.

At execution time, the execution control addresses **again** line by line and examines the presence of these TOKENs. For almost every TOKEN, i · BASIC interpreter has its own execution routine, which **is** called at · Find the corresponding TOKEN to perform the hidden function *behind it*.

These command and command execution routines then perform another formal (syntactic) check:

- is the number of paraneters?
- Are the correct data types used?
- Are the comas in the right place?
- are parameters enclosed in clear if necessary?

uS,

In a copiler, these functions would all be omitted at run time, since they are already in place during the compilation.

For LASER, execution control **is** invoked if a konando or progre has been entered le without a line number, or if a KNN tone sand **has been** detected.

An R-#aando is a complete BASIC-Progra stored in the Prgra table. .

Execution control starts at address 1D1EH and ends at address 1D77H. The insertion occurs at address 1D5AH.

The following steps are performed when running a Progra1111 line:

1. Load the first character of the current row from the Progran table.
   When the end of the programme table is reached, the programme returns to the input routine.

2. If the tracer is on (TRON active), the line u1111Rer is output to the dell screen (nn),

J. If the character is not TOKEN, go to 7.

4. If the character > 'BBH', it must be exactly 'FA}' (MID$), otherwise aa line start is not allowed and it will be created SYNTAX ERROR.

5. If the character is less than BCH, it will be used as an index for a jump table of execution routines.

6. The corresponding execution routine is called and 1 after it is executed. jumped back.

7. If the sign is not TOKEM, it is a value.
   The specified variable is obtained, if it does not exist, will be added to the variable table.

8. The value expression is analysed and the value of the variable is assigned.

9. Back to 1,

Aner kung!

The TRON and TROFF commands are available only via EXTENDED BASIC. However, you can replace them with POKE commands in the normal BASIC.

POKE 31883,1 = TRON POKE
318830 = TROFF

The execution routine starts with the loading of the first character of the line to be edited.

The address 1D1EH is then packed onto the stack. This is the address to which all execution routines should return after successful completion of their operations.

If the read character is not a TOKEN ((IIIIHI), it should be a value111statement, e.g. B=5.

The routine for performing Value111eisations starts with IF21H. This is the same address where you land if the TOKEN 8CH for LET had been available before the assignment. For this reason, the LET can also be omitted.

The lecture rutine expects the adre$pointer to the row to be edited to be immediately in front of the variable names. Searches the variable table for an entry of the same name, if it does not exist, the nane will be added to the table. Behind the variable name must be an equal sign and then the value expression. The value expression is analysed in the routine starting at 2337H. The value of the111is then converted to the correct type of the specified variable and stored at the variable address.

If a TOKEN is detected at the beginning of the line, lllis checked if it is a valid TOKEN. Only TOKEN 88 to BBH are valid at the beginning of a statement, TOKEN ICH to F9H can only be used as part of a value assignment or a command sequence.

Example: 8FH (IF) 'expression' CAI (THEN) xxxx

> The TIEN-TOKEN must not be at the beginning of a line, but only after an IF statement.

The only exception to this is the MIDS-TOKEN 'FA', which can not be used at LASERs Without further, as it uses one of the unused RA extension outputs in the coordination area. Here, however, new, self-made BASIC commands can be cleverly connected to the interpreter, e.g. a SORT or similar.

A TOKEN between lllllH and BllH is used as an index in the jump table from address 1822H, in which the start address of the execution routine is stored for each valid statement. The programme will then continue at the address given there.

The values behind **the** TOKEN until the end of the statement are the parameters required for the execution routine. Each execution routine knows the expected parameters and checks them for completeness and correct format. The end of the parameters of a command nuss aurh is the end of the statement inside,

After completion of an execution routine, the control is passed back to the execution control (1D1EHl), where the end of the statement is checked. The end of a statement is either an end-of-line identifier **Mil** or a statement separator ': · . When a cutting separator is reached, the following statement in the same row is interpreted and executed in the same way.

When the end of line is reached, the next line in the programming table is addressed and passed to execution control.

However, in the case of a ystetando or a direct command, there is no 'next' line, these new instructions are not executed from the programme table, but directly from dE!ll input/output buffer. There is a programming end at the end of the buffer **Mil-Mil** si1Uulated. An additional line number of FFH-FFH or 65535 is inserted to identify such a statement.

The RUN command informs the execution control, since$ has to take its instructions from the Progra table.

When the end of a BASIC programme, whether nit or not, is reached, the EN execution routine jumps back to the input routine.

An error that is detected during execution causes the output of a corresponding error alcl.lng and also the return to the input routine.

The merchandise in

The execution routines .execute the actual functions of each command. For each system command (CL.OAD, CSAVE, LEAR, *RN* etc.) and for each BASIC command IFOR; IF, 60SUB, GOTO, etc.) there is a separate execution routine. In addition, execution routines are available for all athematical functions, such as SIM, CS, ATM, LOG, etc.

These execution routines analyse the statement from the point where the execution control discovered a TOKEN. The statement is examined from left to right for special characters, such as Komas *or* KI amern or TOKEN. Each statement has its own special paraet structure, so that a further formal check takes place here.

In many cases, the execution routines also perform 111 control functions by calling a series of internal subroutines to perform their function, which they share with other execution routines.

A good example of such an internal subroutine is the expression analysis at 2337H. This routine is called by all execution routines that allow expressions in their parameters. Examples of such execution ruts are those for processing IF, FOR and PRINT.

The expression analysis calls further internal subroutines, e.g.11. 268DH u111 to edit variable within an expression. Since there are also indexed variables that allow an expression as indexing, this routine must call back the expression analysis subroutine that just called it itself. This is called recursion.

An example of a value assignment that causes such recursion:

ST = ESC/CD(3,F)/D(EN-1)

Other internal subroutines are:

- Send to the end of instruction <1FB5HI
- Find a FOR or GOSUB data block on **the** stack (193#)
- Write an entry in the string cache (2865H)

and many more.

Intermediate results are usually placed in **a** working area of the Coordination Area <X-Register = 791DHI.

All execution routines (au&er MID$) are sprinkled with the following register contents:

A -	the following TOKEN sign CARRY = nuer isrh
Flags BC	ZERO = End of line or end of line '@' Start
- DE -	address of execution routine
II. -	Address of the + 1 jump table entry for the TOKEN address of the character in the statement in A

A table of all system commands and BASIC commands with *the* address *of* their execution routines is included in Chapter 9.

ri aesthetic and **aatheatistic** functions

First, some reflections *of the* computational abilities of the ZBI and the BASIC interpreter.

The Z80 supports internally only B bit and **1** bit additions and subtractions. He does not perform 11 multiplications or divisions, and even less does not perform fluidized arithmetic.

The register set of the Z88 for *the* implementation *of the arithmetic consists of seven 1-bit register pairs* (A, B, DE, H, I, I, SP), plus *four* shadow register pairs (AF', BC', DE', HL'l, which can only *be used* for intermediate storage.

All arithmetic operations must generally take place between these registers. Register storage operations are only possible in a few exceptional cases via indirect addressing.

The operations *of the* registers are also limited. Especially in *the* 16-bit arithmetic only very best register constellations are allowed.

The BASIC *interpreter* supports all arithmetic operations, whether addition, subtraction, 11 multiplication or division, and this for three different types of variables:

- Integer Variable (Integer)
- single precisionl **variable**
— Variable double precision

This **is** achieved by internal subroutines that replace the missing *hardware capabilities* **of the Z88** with software.

However, due to how complex *the* software is, no ash operations are supported, i.e., within only variables of the same type can be linked together. Using unequal types of variables in an operation **would** produce unpredictable results, creating a new type of random generator.

The three types of variables supported by the BASIC interpreter have the following format:

Integer variable:	16 bits (1 bit sign, 15 bits data) 32 bits
Simple precision variable:	(8 bits exponent, 24 bits nantisse 11it sign)
Double precision variable:	5 bits (8 bits exponent, 48 bits mantissa 11it sign)

This will show that the hardware registers are not sufficient to acquire, let alone process, two variables of simple or double accuracy.

For this reason, two working areas have been set up in the communication area, which are used as intermediate and working memory (register replacement),

These are the 1 workspace (called the register), which occupies the range 791DH to 7924H and *the* work area 2 (Y-register), which extends from 7927H to 792EH.

These two areas of work have the following format:

Address	Integer	Simple accuracy	Double accuracy
791D			LS.B
791E			NSB
791F			NSB
7920			NSE
7921	LS	LSB	NS.B
7922	tSB	NSB	NSB
7923		NSB	I'ISB
7924		EXP	EXP

(Addresses refer to workspace 1, but workspace 2 is structured the same)

LSB = least significant byte NSB = next significant byteJ
MS} = most significant byte almost significant byte)
EXP = Exponent

The various arithmetic operations for the three variable types have the following register/workspace mapping:

Integer variable

1.Operand		2.Operand	Operation	Result	Run routine
HI...	%	EN	addition	HI...	0BD2H
HI...		EN	subtraction	HI...	@RC7H
HI	%	EN	1'1multiplication	HI...	0BF2H
EN	/	HI...	Division	ARB1	24901-1

Simple precision variable

1,Operand		2.0perand	Operation	Result	Run.routine
ARB1	+	BCDE	addition	ARB!	071H#
ARB1	-	BCDE	subtraction	ARB!	0713H
ARB!	%	BCDE	l'lultiplication	ARB!	0847H
ARB1	/	BCDE	Division	AR11	08A2H
				1	

The two register pairs BC and DE are used for operations of simple accuracy to display the 2. Operands.

double precision variable

!,operand		2. operand	Operation	Result	Exp. routine
AR11	+	ARB2	addition	ARB!	0C77H
AR111	-	ARB2	subtraction	ARB!	0C71H
ARB!	*	AR112	multiplication	ARB!	DA1H
AR111	/	ARB2	Division	AR111	0DE5H

As gelüschte operations are not allowed, integer values can only be processed with other integer values. The same applies, of course, to values of simple or double precision.

Since four different arithmetic operations are possible for each type + - * /1 and there are three different data types, S011it also has twelve different arithmetic routines. 3 routines for type-dependent mathematical comparison operations were added.

An address table of the 15 routines is included in the ROl'I from address 1BABH.

Each of these returns knows the type of values to be processed and expects$ to be available in the correct registers or ranges.

This is not true for the mathenatic routines, since they only work with an input value, which must be provided within the workspace 1 {I-Register).

Now there is a problem for the mathematical routines. You have to call arithmetic operations internally, but you can't see the type of the argument provided in workspace 1. For this reason, another byte (78AFH>) was used in the communication area, which gives information about the type of the value stored in workspace 1. This byte is also called a type flag.

It shall contain one of the four following codes:

Code	Data type
82	Integer text variable
03	simple precision
04	double accuracy
08	

The type code is exactly the length of the value stored in workspace 1 for the specific data type.

The athematical routines allow all but a few exceptions to all the different data types (see detailed description of the routines).

Internal display of data

For a better understanding of the previously stated 111erden here, • how the data in LASER are actually presented internally.

Integer variables are represented in 1 bits, with bit 15 containing the word mark and bits III - 14 containing the value. The resulting maximum representable positive value is 32767 (decimal) or 7FFF (hexadecimal). The smallest representable negative value is -32768 {decimal) or 8808 (hexadecimal).

bit=> 15 14 13 12 11 10 9 7 5 4 3 2 1 0

sign @ = Valu
positive 1 = e
negative

positive values: **0000** - 7FFF (hex)= **0** - 32767 (decimal
negative values: FFFF - **8000** (hex)= -1 - -32768 (dec)

Note that negative values are represented in their ler complement.

Flow coma variables are processed by BASIC in two different types:

- Simple precision variable
- Double precision variable

For types, there is an eight-byte sign exponent.

For simple precision variables, the mantissa has a length of 24 bytes with a sign and a length of 56 bytes with a sign for double precision variables.

Both types have the same shape, they differ only in the 11antissenlength.

Bit => 31 24 23 16 15 B 7 @

sign of the exponent 1 = positive (the **coma nut** to the right) **0** = negative (the **Kanna must** go left>

sign of the Atlantisse = positive 1 = negative

value of the exponent = Number of digits to move the kona.

value of the ltantisse, left-justified (normalised), i.e. the highest bit value is in BitPosition 23 • Negative values **are** represented **as** positive, **is the** only difference because the bit 23 is set.

Type= simple precision

The number representation takes the form number= l'lantisse * 2 EXP ait

0.5 (= ltantisse < 1

The nantisse is 2 or 5 bits long; where the first bit is not ・it stored, because it is i1111er 1. In its place is the sign of the Nantisse.

The exponent is stored i1111er ait a offset of 128 (= BIH), from this the sign of the exponent in the highest-value Bi posi tion.

Examples: 8.5 = 0.5+20 == Ex = O, N= 00 08 00 -4 = -0.5+2S ==> Ex = 3, K= 8808 8
-0.25= -0,5 +2 -1 ==)> Exp = TF, N= 8 0111

The greyed number, which can be displayed with all its digits in simple precision, is 224-1 or 8388687 (dez,l or 7FFFFF (hex.), With double accuracy With the greater mantissa, the largest exactly representable number is 23-4 or 3,57811l6 {dez.) or 7FFFFF FFFFFFFFFF (hex).

However, these values, 838807 or 3,578w40l, do not represent the greatest value, ai t dea can be calculated, but only the grey one, which is representable without loss of accuracy, that is because the exponent for both types can assume values between 2 -128 and 2 127 In theory, the Koaa! in binary representation) can be pushed 127 places to the right or 128 places to the left, although only 24 and 56 bits can stand in the ltantisse.

Depending on the type of data and the calculation, this is also ■ sufficient. Only the number of valid digits of a value is decisive.

er kungG

With LASER 111, 210, 318 and the VZ200 1ä$t, it is not easy to work with double-precision variables. In the basic version of the BASICInterpreter, you can only choose between simple precision variables and integer variables.

- Variable names without addition = simple accuracy -
- Variable name ait Add 't' = integer variable -

However, there is a little trick by defining yourself by a POKE xxxx,B8 in the type table of the configuration area (from 7981Hl onwards all besti・te variable names to double accuracy.

Egg:/ Output driver

Drivers have the task of creating an interface between the delll logical concerns of an application (a programme) and the physical conditions of a specific input/output device.

For example:

A programme would like to put a very best-inked character, e.g. the letter 'A', on the cassette. This character is provided in the ASCII code in the ZBill's A register and the driver is called. The driver takes the character and transfers it to the cassette output {bit 1,2 of 680MH address, bit by bit, serially, in the pulse sequence specified by the recording method.

I・ LASER collectors are available with drivers for the keyboard, monitor, parallel printer and cartridge.

Keyboard and printer drivers are addressed via a special device control block (DCB) at 7815H and 7825H respectively. There is also a call DCB at 7B1DH for the screen, but LASER does not use it except the cursor management.

In the DCBs, the counter and pointer are held for the special device, e.g. paper eel and line counter for the printer or the cursor position for the screen. Furthermore, the driver addresses can be found there.

The DCBs are set up in the system initialisation via the communication area and filled with default values.

A driver is called for each character to be transferred. Drivers do not know records or files, they cannot block or unblock characters. Such functions shall be performed by the calling programme itself. MAI interpreters usually do this. the routines for PRINT and INPUT.

When writing to the cassette, the PRINT-Koando first creates a header of 255 18111Hund51 FEH, the data identifier F2H and the files. Then each variable is transferred as an ASCII string. Kc:Mas separates the individual variables and a carriage return character (CR) completes the transfer.

An INPUT-Koaando first searches for the header, checks data identifiers and FilenameMen, all variables are then put in the input/output buffer over one after the other, until a carriage return character (CR= BDH> is discovered.

Then **each** variable **is converted back** to the correct format and transferred to the variable table.

The keyboard driver starts at 2EF4H and extends to the address **Ik,** including an ECHO Rutine that represents the characters entered directly on the screen. In addition there is a routine from 0507H, which handles the operation of several buttons simultaneously, the so-called 'ROLLOVER' .

The keyboard driver can be called directly from the Device Control Block (DCB) with CALL 2BH when a single character is to be picked up from the keyboard. For example, this is the routine of the BASIC command INKEV$, but this type of query does not involve an ECHO output to the screen.

However, keyboard query is also integrated in *the* interrupt service routine {see Screen Driver Description). This type of query is mainly used when an entire line is read in via the screen editor at 3E3H and the entered text should be switched to **the** screen.

The screen driver extends from 3039 to 342FH and consists of various sub-routines.

A particular problem must be taken into account. The VIDEO-RAN **is** accessed by two different modules. On *the* one hand *the* image generator constantly scans the VIDEO-RAN and transmits the characters contained there to the screen, on the other hand the information **must** be written by Z88 into the VIDEO-RAN. If this happens unsynchronised next to each other, you get an unpleasant flickering screen.

This is picked up by spending on the VIDEO-RAN only in the dark phase of the screen. This process is controlled and synchronised via the image generator's vertical synchronisation signal. This is used to create an interrupt. Screen output is first stored in a special *putter* from 7AAFH and only in the interrupt service routine, i.e., during the dark phase of the screen11S into VIDEO-RAi'! .

The interrupt service routine is at 2EBSH. It also leads besides the buffered screen output described above (via 3F7BH to 38EH) additionally

– the blinking cursor representation (2EDCH1,
– keyboard query (2EFIIH1 and
– ECHO's edition on the 11 (301BH1)
a beep sound (33MH)

dialogue box.

The interrupt is performed every 2 til liseconds on PAL system. Prior to the execution of the above Functions are jumped over the RAN 787DH extension output, a good opportunity to work even 11it dell Interrupt, e.g. to implement a software clock.

To manage the screen, there is a table at the end of the BASIC Coordination Area (from address 7AD7H1 onwards, which gives information about the status of each line in 1-byte entries.

88H - line is a single line of 32 characters
81H - *row* is first of a double line of 64 characters
H – row is second of a double line

This table in conjunction with two status flags at 7838H and 7839H are the basis of the screen editor at 03E3H.

The printer driver starts at 858DH and continues at 3ABH. This is the noise when a normal ASCII character is to be output and the driver is called via the DCB.

A special type of output is the COPY-Betehl (from 3912Hli with the help of which the screen content is printed out completely. If you have *a* printer of the brand Seikosha GP1, or similar, also inverted characters (transfer table starting from 39M4H), block graphics and even images of the high resolution graphics can be printed out!The colours are replaced by different shades of grey.

A 'Seikosha GP1@@' Capable Printer Must meet the following conditions:

– Switch Code Text=> Graphics = @8H
– switch code Graphics=> Text = 0FH
– Single needle control for 7 needles

The cassette driver extends from 34A9H to 389CH and consists of many individual rootins, which are called depending on whether a cassette input or output occurs, who the

The cassette recordings shall take the following format:

1. BASIC and NaschinenprogralMII!

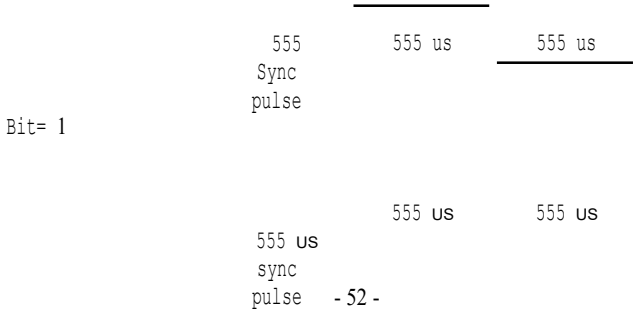
255x 811H	synchronisation byte
5 x FEH	
1 Byte	Pr. Recognition 00 = BASIC programme @1 = Machine programme
Max 15 bytes	Pr. Name
liifllH	Name Field End Id
2 Byte	programme start address
2 ,	End address of the programme + 1
n Byte	programme text
2 ,	Pr Gfsunne
20 x purple	End Id

2, data files (from BASIC-PrograMl!ll with PRINTll

255 X 80H 5 x FEH 112H max	synchronisation byte
15 bytes	ID for a data file file name
0eH n bytes	End Id for Nallensfeld String variable, coma separated End Id
IIDH	(Carriage Return)

The tape recorder is bitserial in the following beep:

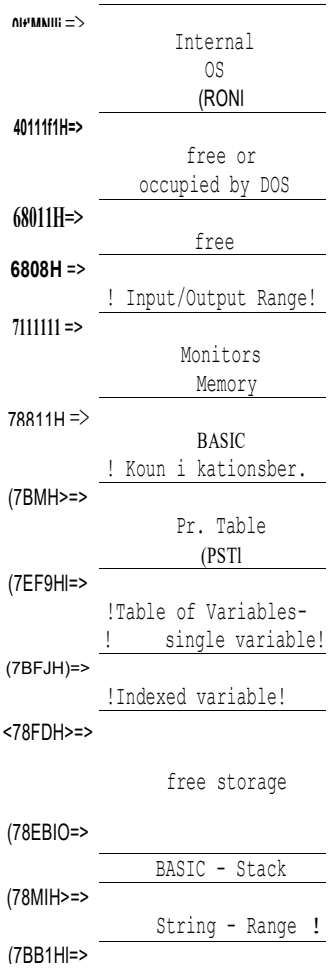
bit = 8



First, a 555 "microsecond synchronisation pulse **is** output. A binary mull has a long pulse of a total of 1110 "microseconds, and a binary one has two pulses of 555 microseconds each. This results in a total pulse length *for* a bit recording of 1665 l'likrosecond and
speaks to a recording rate of Bit/sec (Baud).

The messages generated by the cassette driver rutine in the last screen line can be suppressed if necessary. You can do this at 784CH.

784CH	=	Notifications are issued
784H4}**8**		Messages are suppressed.



nnnn = absolute location of range i ' Memory
(man) = **Adre$zeiger** in Kounikat in range

Internal tables

Internal Tables are the lists and tables that are fixed to the RN of the operating system. At the same time, this means that the content and location of these tables are fixed. It is used by BASIC interpreter for syntactic checking, expression analysis, data conversion, and execution of best inter commands (e.g., FOR, IF).

JASIC-5 Key Matrix (1650H-1821H)

This table contains all the *reserved* words and characters of the BASIC interpreter, be they instructions or functions.

Each entry contains a word, where the first character of the entry is 2l=1.

During *the* input phase, a read-in line of characters for characters 111it of this table is compared, if a text section *of the* line matches an entry, it is replaced by the so-called TOKEN. A TKEN is a 1-byte hexadecimal value, *which is* derived from *the* lfd. Entry reason found. In addition, Bit 2/ is set.

Example'

CLS is *the* 5th. Entry in *the* table.
Since B is started to count, this results in an entry num of 4, If turned on with 2, a TOKEN of 4 fr CIS results.
I.e. each string CLS i programme text is replaced by 84H, with the line length shortened by approximately 2 bytes.

If you examine the table, you will find *that$* is a set of keywords not encoded but composed of binary zeros. These are included in *the* list below and marked with an asterisk, except for DOS commands, but the execution routines are largely present.

M.E. is *the* reason to look in copyright to make one too grudge! Avoid resemblance to another computer. With EXlc.NDED BASIC, many of these commands are made available to the user.

keyword	TOKEN	keyword	TOKEN	keyword	TOKEN
END	**8Wed**	(NAI'IE)	A9H	OR	DJH
FOR	81H	(KILL>	AAH M	>	D4H
RESET	82H	(LSET1	AB	=	DSH
SET	8YH	IRSET>	ACH 1	<	DH
a..s	BA	(SAVE)	ADH 1	S6N	D7H
(CND1	85H *	(SYSTEM)	AH 1	INT	D8H
(RANDOf11	**86H** *	LPRINT	AFH	**ABS**	D9H
NEXT	87H	(DEF1	**BIH** *	(FREE	DAH *
DATA	**B8H**	POKE	B1H	**INP**	DBH
INPUT	drop	PRINT	B2H	(POS)	DCH
DIN	**SAH**	CONT	BJH	SQR	DCH
READ	SBH	LIST	B4H	**RND**	DEH
LET	SCH	LUST	B5H	LOG	DFH
60TO	SDH	(EN1ET1	**B6H f**	EXP	EIIH
RUN	**BEH**	(CAR)	B7H *	COS	E1H
IF	**BFH**	a..EAR	B8H	SIN	E2H
RESTORE	**98**	CL.0	B9	TAN	E3H
605\JB	91H	CSAVE	BAH	ATN	E4H
RETURN	92H	**NEW**	BH	PEEK	E5H
REN	93H	TAB	BCH	ICVI1	EH
STOP	94H	TO	BDH	ICVS1	E7H f
ELSE	95H	(FN)	BEH *	ICVD)	DAB
COPY	96H	**USING**	BFH	IEOF)	E9H
COLOUR	97H	(VARPTR)	**CM# +**	(LOC1	EA}
YERIFY	98H	USR	C1H	(LOF)	DAB *
(JEFINT1	99H *	<ERL>	C2H *	**(I'IKIS)**	ECH
<DEFSNG>	9A} ±	(ERR)	C3H *	(NKS$)	EDH *
. (DEFDB1.1	9B} M	(TRINS$9)	C4H *	($11KD)	EEH *
CRUN	9CH	(INSTR>	C5H *	<CINT)	EFH
IODE	9DH	POINT	CbH	(CSNG1	FOH +
SOUND	9EH	(TINE$)	C7H *	(CDBl.l	FIH
(RESUE)	9FH *	(EM)	C8H *	(FIX)	F2H
OUT	**AN#**	**INKEYS**	C9H	LEN	FJH
(ON1	AIH *	TIEN	CAH	STR$	F4H
(OPEN)	A2H *	NOT	CBH	VAL!	F5H
(FIELD)	A3H *	STEP	CCH	ASC	FbH
(6ET)	MH *	+	CDH	CHR$	F7H
(PUT>	A5} +		CEH	LEFTS	FBH
(Cl.OSE1	A6H *	%	CFH	RIGHTS	F9H
(L.0A)	A7H *	*I*	111H	**MID$**	FN
					FJH

I1 ROI1 exists two different AdreStabel lines for the execution routines. The first **is** used by execution control when a BASIC statement is to be executed. It contains addresses for the execution routines of TOKEN 81H to **BBH** and is located in adress range 1822H - 1899H. The first TOKEN of a statement (Bit7=) is used as an index ({0 - 59) for table access. From the table, the address of the execution routine is copied and called. If the statement does not start with a 111 TOKEN, the routine tGr value assignments is branched (implicit LET),

The **second** address table of 1H - 1FM contains the addresses of routines for BASIC functions, which may occur only on the right side of a value zero!

If during expression analysis a TOKEN i11 range D7H to FAH **is** encountered, it **is** used as an index (0 - 35) for the second AdreOtable and the address of the execution routine is taken from there,

For the TOKEN bra to Dl! no address table is required because this is Up-occur directly from the other execution routines.

BASIC Statement Address Table (1822H - 1899H)

TOKEN	keyword	Address	TOKEN	keyword	Address
88	END	1DAE	9E	SOUND	2BFS
81	FOR	1CA1	9F	RESUI1E	1FAF
82	RESET	**8138**	*A*	OUT	2AFB
83	SET	0135	AI	ON	1F6C
4	a..S	019	A2.	OPEN	7979
85	CND	7973	A3	FIELD	797C
8	**RANDOfl**	81D3	A4	GET	797F
87	NEXT	*22Bb*	45	PUT	7982
88	DATA	1F85	**Down**	Q.OSE	7985
89	INPUT	219A4	A7	LOAD	7988
SA	**Dil'f**	26088	AS	IERGE	798B
8"	(REAi)	21EF	A49	**NAIE**	798E
SC	LET	1F21	**AA**	KILL	7991
8D	GOTO	1EC2	AB	LSET	7997
BE	RUN	lEAJ	AC	RSET	799A

TOKEN	keyword	Address	TOKEN	keyword	Address
8F	IF	2039	AD	SAVE	7948
99	RESTORE	1D91	AU	SYSTEM	0000
91	605\JB	1EB1	A	LPRINT	2067
92	RETURN	1EDE	B	IIEF	795B
93	REN	1F07	B1	POKE	2CB1
94	STOP	1D9	B2	PRINT	206F
95	ELSE	1FIJ7	B3	CONT	1DE4
9	COPY	3912	B4	LIST	2B2E
97	COLOUR	389D	B5	LLIST	2R.29
98	VERIFY	3738	B	IIELETE	2BCi,
99	IIEFINT	1E03	B7	CAR	20IIJB
9A	DEFSNG	1E86	B8	CLEAR	1E7A
9B	IIEFDBL	1E09	B9	Q.OAD	35
9C	CRUN	372E	BA	CSAYE	34A9
9D	NOIIE	2E3	BB	NEW	1B49

MSIC Function Address Table (1688-164F)

TOKEN	keyword	Address	TOKEN	keyword	Address
D7	SfiN	0984	E9	EOF	7961
D8	INT	IJB37	EA	LOC	7964
D9	ABS	9977	EB	LOF	7967
DA	FRE	2704	EC	NI$	796A
Profit	INP	2IE.F	ED	NS$	7960
DC	POS	27F5	EE	$11<1)	7978
DD	SQR	13E7	EF	CINT	7F
EN	RND	14C9	FO	CSN6	04B1
DF	LOG	1111119	F1	CDBL	M4DB
EO	EXP	1439	F2	FIX	IB26
E1	COS	1541	F3	LEN	2AI3
E2	SIN	1547	F4	STR$	2836
E3	TAN	158	F5	VAL	2AC5
E4	ATN	15BD	F6	ASC	2AF
E5	PEEK	2CM	F7	CHR	2A1F
E6	CVI	7952	FS	1EFT$	2A61
E7	CVS	7958	F9	RIGHT$	2A91
IT	CVD	795E	FA	MID$	2A9A

order of precedence of operations

The ranking of the various arithmetic operations in arithmetic expressions 111is also given with Hi 1 fe of a table **er1li** which is located **in the** address range 189AH to 18A0H.

This table contains numeric values for the different operators that define the precedence.

In expression analysis, each operator/operand pair is placed on the stack plus the rank value of the previous operator. If an operator with a higher rank than the previous one is found, the operation is executed immediately and the resulting intermediate result is packed onto the stack.

Operator	Function	rank value
+	addition	79
	subtraction	79
*	l'lultiplication	7C
	Division	7C
/	potentiation	7F
[	logical link	5111
AND	logical link	46
OR		

Mi Aesthetic Routines

For the three different types of new variables, the RON at 18ABH-18CSH contains three tables with the initial addresses of the corresponding aristetic routines. This one uses the expression analysis.

Function	Integer Variable	Variable uningexic.	Variable Dopp.
addition	8BD2	0716	0C77
subtract iDII	8BC7	0713	C70
multiplication	8BF2	0847	9DA41
Division	24911	0842	0DE5
comparison	8439	IIAIC	0478

For completeness, the address of the routine for adding text variables (strings). This is 298FH.

Data - transformation (type matching)

To convert data into the various types of variables, there is **another** table that contains the addresses of the corresponding execution routines, depending on the target type of 111. These routines convert the value in workspace 1 (-register) to the desired data type. They are used 11th of the expression analysis to be able to link values and intermediate results of different types.

The table is at 18A1H-18AAH.

Treatment in	Address
text variable	0AF4 *
Integer variable Variable	0A7F
one.Accuracy Variable	04%1
doppAccuracy	04DM

* For text variable, the specified routine does not include urine conversion, but
just a test to see if the variable in the i111 workspace 1 is actually a text variable, If not, 'TYPE NISNATCH ERROR' will be output.

I ROk-Area have two tables with error messages.

The first table is at 1BC91HBF6H and contains only one two-character abbreviation per error message. It is a relic from another operating system operating system and is not used via LASER.

The table used by the LASER 118, 210, 318 and the VZ2808 is 3CECH-3E2BH and contains the error messages in written-out form.

The error messages are determined by an error code that is used as the index for table access.

Error now	error code	Error Message (Long)
0	NF	NEXT WITHOUT FOR
2	SN	SYNTAX ERROR
4	G	RET'N WITHOUT GOSUB
6	OD	OUT OF DATA
8	FC	FUNCTION CODE ERROR
A	0V	OVERFLOW
C	ON	OUT OF NENORY
E	IJ...	UNDEF'D STATEl'ENT
1Ir'I	B	BAD SUBSCRIPT
12	DD	REDl11'D ARRAY
14	%/	DIVISION BY ZERO
1	ID	ILLEGAi.. DIRECT
18	d1	TYPE MISNA T CH
1A	OS	OUT OF SPACE
1C	LS	STRING TOO LONG
1E	ST	FORNUL.A TOO CONPLEX
28	CN	CAN' T CONTINE
22	NO	NO RESUNE
24	RW	RESUNE WITHOUT ERROR
2	UE	UNPRINTABLE ERROR
28	NO	11ISSIN6 OPERAND
2A	FD	IIAD FILE DATA
2C	LJ	DISK COIHAND ERROR

External table

External tables are the data structures created and managed by van BASICInterpreter in RA} scope. #lerkmal of the external tables is that their content can change and also partially their location in the RAN area.

For tables that change their location in memory, there are address pointers in the BASICounication area so that they can be found and addressed 1I1 at any time.

The BASIC - KoaunikatiOIISrange (7B800H- 7AEBH1)

Ia Communication Area is created and managed by the BASIC interpreter all necessary **address** and management tables that can change during normal programme editing or where the user can define or change **his own** flow variable, if necessary.

Consider the communications pane the BASIC interpreter's notepad.

The following list contains a description of each byte in this area.

Some tables within the notification area are described in detail after this collection.

The range 781111H to 7835H **is pre-assigned when** the Systa initialisation **from delt** ROIt range.

78111	C3 96 1C	JP 1C96H	;RST 8 -
7883	C3 78 1D	JP 1D7BH	Vector ;RST 18 -
7806	C3 **91** lC	JP 1C9OH	Vector ;RST 18 -
7819	C3 D9 25	JP 25D9H	Vector ;RST 28 -
780C	C911i1 II	RET	Vector ;RST 28 -
788F	C9 III	RET	Vector ;RST 3B -
11i1		EI	Vector ;RST 38 -
7812	FB	RET	Vector
7813	C9 1Ii1		

Keyboard Device Control Block (DCB)

7815	81	DCB+<enner
7816	FA4 2	Driver Address
7818	**88**	Background flag
7819	**8o**	(l=green, !=black) All right. Background
781A	**88** 7813	
4B 49		'AI'

Monitors Device Control Block <DCBJ li
• LASER 118-318 unused> DCB identifier
(deleted>
Pointer to start-of-programme
address on CLOAD
cursor address

VerifSUMe for Cassettes Input/Output

Printer Device Control Block (IDCB)

DCB+<enner
Driver Address
Lines/Page+1
line counter

'PR'

781D	**88**	
781E	**080 00**	JP 5811I8H
7820	00 70	
7822	**88**	
7823	**80.08**	Buffer B1 for **1**. Keystroke code while pressing repeatedly Buffer B2 for 2. Keystroke code while pressing repeatedly
7825	**86**	RST 8
7826	**80.05**	
7828	43	LD **A**,8
7829	**88**	RET
782A	**88**	
7822	50.52	
782D	C3 **88 58**	
7838	C7 **08 00**	
7833	3E 08	
7835	C9	
7836		
7837		

;unused ;u
nused

; if DCB+<ID A=0 is unknown

7838	FLAG 1 Bit 7 - CONTROL-Flag Bit 6 – REPEAT-Flag Bit 5 - WAITFIag Bit 4 - B2 status flag Bit 3 - Bt status flag Bit 2 - FUNCTION flag Bit 1 - INIIERSE flag Bit a - SHIFT flag
7839	FLAG 2 Bit 7 – unused Bit 6 - CRUN flag Bit 5 - Ini-Flag f. buffered output bit 4 – flag t. INPUT Statement Bit 3 – VERIFY-FIag Bit 2 - BREAK flag Bit 1 - BUZZER flag With - Carriage Return Flag
783A	time counter
7831	INPUT/OUTPUT Latch
783C	Character backup for cursor display
783D-7841	unused,zt
7841 7842-	flash counter
7843	Keyboard Query Cache (Row/Column)
7844-7845	Keyboard Query Cache (Olatr i x Address)
	colour code
784 7847-	unused
784B 784C	Output Flag t. Field output on cartridges I/0 ()8 - Messages who suppresses the)
784D-787C	unused

787D	C9 1110 1110	RA} extension output of interrupt service routine
		The range 7880H-7BA5H **is** initialised from
		dell RON range filled
		Subprogramme ' for division
7880	D 0e	SUB 0 ;Subtraction 22 - Z1
7882	6f	LD L,A ;10 **is** specified before each call.
7883	7C	LD A, H
7884	EN 1110	SBC **+0**
7886	67	LD H,A
7BB7	78	LD M,B
7888	EN 1110	SBC A,0
7BBA	47	LD B,A
7BBB	PER 1110	LD A, 0
788D	c9	RET
7BBE	4A 1E	USR - Start address
		pre-loaded with FUNCTION-CODE Error
7898	40 E 4D	1'1Ultiplikator f. RND
		Subprogramme for INP
7893	DR 08	IN A,(0)
7895	c9	RET
		Out Subprogrammes
7896	D3 08	OUT (@),A
7899	110	INKEYS Cache
789A	080	ERR last error code
789B	110	Printer Location in Row
789C	%0	Device Flag (8:Image, !=Printer, 88=Cassette)
789D	4	Line length on **dell** Monitors preloaded with)
789E	38	last tab position (pre-filled with 48)
7B9F	8%o	unused
7BA8	47 7B	Initial address of the string range

current line number

7BA2 FE FF

Initial address of the programme text

7844 E9 7A

Output Image Column Pointer

7BA6-7BA7

Pointer to input/output buffer (from 79EBH)

7BA7-7BA8

Input flag (0 = cassette)

7849 7BM-7BAD

last random number

7BAE

DIN Statement Flag

7BAF

Type of value in !-Register @2
= Integer
3 = string
4 = single precision 8=
double accuracy

7BB8

Intercode Generation Fla for DATA Operation
Code at *the* Expression Analysis

78B1-7BB2

End address of the BASIC memory area

78R3-78B4

Pointer to string cache String cache

78BS-78D2

(10 x 3 bytes)
(1 byte - length, 2 bytes - address i111 string range

1. string cache

78D3-78DS

7BD6-78D7

Pointer to last free byte in string area Allg,

7809-7809

address cache
Fornatflag t. String output of a number

DATA - Line Number

78DA-78DB

flag blocking *the* indexing

7BDC

RESIIE/RETURN flag

78DD

78EN	Intermediate buffer for Gr PRINT USING DATA flag for INPUT and others.
78DF-78E0	generic address memory e.g. **NE** FOR/NEXT run variable continuation Number of variables table at LET
78E1	AUTO Input - Flag (0 - no CAR>
78E2-78EJ	JT0 Line Number
78E4-78E5	AUTO - Increase value
78E6-78E7	Address *of the* current line (FFFF = direct command)
78E8-78E9	Pointer to the BASIC stack
78EA-78EB	The line where *the* last error occurred, the **pattern**
78EC-7BED	*of the* line where *the* last error occurred (,-option on LIST>
78EE-7BEF	Address *of the* line where *the* error occurred Address of an error handling routine (ON ERROR) Error -
78Fl-78Fl	Flag (Error=255, RESUME=0)
78F2	Address of the decimal point ia pressure buffer
78FJ-78F4	line *number*, *which* was last interrupted (IEND,
78FS-	STOP, BREAK)
7IIF6	Address *of the* line *where* the last break occurred
7IIF7- 7IIF8	Progra. End Address Start *of* variable table End address of 1. Part *of the* variable table beginning *of* Natrix table 12. Part)
7IIF9- 78FA	

7IIFB- 7IIFC	

7BFD-711FE	Initial address of free memory (behind the Natrix table)
7BFF-79111	pointer to DATA row

Type Code Table

7901	A
7982	B
7983	C
7904	D
7905	E
79%	F
7917	G
7988	H
79ffl	I
7904	J
798B	K
798C	L
791D	11
790	N
791F	O
7918	p
7911	Q
7912	R
7913	S
7914	T
7915	U
791	V
7917	W
7918	X
7919	V
791A	Or
7911	TRACE FLAG (= TRON, AF = TROFF)

X - Register

791C zus. Move L.lyte Right

	INT	STRING	SINGLE	DOUBLE
791D				LSB
791E				LSB
791F				LSB
7920				LSB
7921	LSB	ADR	LSB	LSB
79'Z1.	l'ISB	ADR	LSB	LSB
7923.			l'ISB	MSB
792'+			EXP	EXP

7925

Cache for arithmetic operations. e.g.

7926-792E V-Register
(splitting as X-Register)

unused
792

pressure buffer
7939-7949

Additional Multiculation and Dual Precision
7944-7951 Registry

IWI-Vectors for floppy commands
preloaded nit 'JP 112DH' (DISK-CO}RAND - Error)

CVI Statement
7952 FN Statement
7955 CVS Statement
7958 DEF Statement
795B CVD-Anwi 5II9
795E EOF Statement
7961 Lot Statement
794 LOF Statement
7967 #AIt Statement
796A l'I<SH Statement
796D NKD$ Statement
7978 CQ-Anwe i sung
7973 T]NE$ Statement
797 OPEN Statement
7979

797C	FIELD Statement
797F	GET Statement
7982	PAJT Statement
798	CLOSE Statement
5	as in the LOAD
7988	Statement NERGE
798B	Statement NA}E
798E	Statement KILL
7991	Statement
7994	k - Statement
7997	LSET Statement
7994	RSET Statement
799D	INST R
79	Statement SAVE-
49	How To
79A3	L INE Statement
	RAN expansion outputs pre-
	populated With C9-OMH-OH (RET)
794	from ERROR routine
79A9	from USR routine
79AC	Start BASIC loop
79AF-79BI	unused
79B2	from programme at the
7915	input end of
79B8	programme input end
79BB	of programme input
79BE	from NEW and END
79CI	final query PRINT
79C4	data output read by
79C7	keyboard RUN
79CA	execution
79CD	To top PRINT Statement
79D8	PRINT Statement PRINT
79D3	Statement PRINT
79D	Statement PRINT
7909	Statement INPUT
79DC	Statement
79DF	NID$ as INPUT
79E2-79E4	Statement
79E5 3A 00 2C	READ + INPUT + LIST unused
	1/0-Yellow-butter spanner

Input/Output Buffer

79B8-749€

BASIC Stack During Initialisation

79F8 7A9D-

Programme On/File Name - Cassette In/Out

7MD

Cache

Column display on **dell** screen Additional

7AAE

output buffer for buffered image output

Number of characters ia

Putter Buffer-Pointer

7AF 7AB1-

4 byte buffer for graphic printing, SOUND and so on.

7AB1

Cassette I/O Counter f, o,a,Putfer + Length **Name on**

7AB2-7AD1

Cassettes I/O

7AD2-7A5

7AD

Line Status for Screen Rows (9Single Line,
81=Double Line, Subsequent Line) Row 1

Row 2

Row 3

Row 4

Row 5

7AD7 Row 6

7ADB Row 7

7A09 Row 8

7ADA Row 9

7ADB Row 10

7ADC Row 11

7ADD Row 12

7ADf. Row 13

7ADF Row 14

7NE Row 15

7AE1 Row 1

71E.2 unused

7AE3 Pr ogran - At Beginning

7AE4

7NE5

7NE

71E.7

7AEB

This is a table within the communication area. It is used by BASIC interpreters to cache text variables (strings) **that** occur **in** string additions or some PRINT operations.

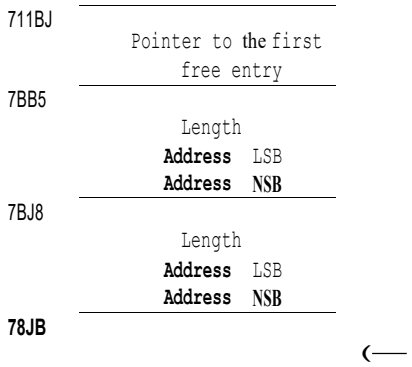
The table consists of ten 3-byte entries, which are consecutively drawn. **All** the top of the table, at 78BJH, is an address$pointer to the next free entry. When the computer is initialised, it is set to the first entry of the table.

Each entry consists of a length byte and an address pointer to the string **in the** string pane or in the programme table. Entries are **assigned** from top to bottom and **then** shared from bottom to top.

If the table overflows, the error message

FOIIU TOO cmux

output.



This table **is** used to **voii** BASIC interpreter and to determine the data type of a variable (integer, String+ united, precision, dopp. accuracy).

The location of this table is also fixed in the communication area, so no address pointer is required for addressing. The content can be changed; **for nor11ale11** BASIC 11it Help the POKE command, in EXTENDED BASIC with the DEFxxx declarations.

The table is 26 bytes long, which corresponds to the number of characters of the English alphabet. There is one byte for every letter from 'A' to 'Z'. The index for table access **is** the first letter of a variable name. *Each* entry contains a code that provides information about the agreed variable type:

82 - integer variable
03 Text Variable (String)
@4 Simple precision variable
case8 Double precision variable

When *the* system initializes, all entries are marked **nit** 0, i.e. as a variable of simple precision.

If a variable name already contains a type identifier (e.g. AS or Ball), this priority is given to a table entry to the contrary.

Address	letter	Initialisation type
79111	A	04
7902	B	14
7983	C	4
7994	D	04
7985	E	14
790	F	4
7987	6	14
79011	H	M4
7989	I	14
7904	J	4
798B	K	14

€790	L	4
790	"	04
790	N	4
798F	0	04
7911	p	14
7911	Q	4
7912	R	84
7913	S	84
7914	T	4
7915	U	04
7916	V	84
7917	w	04
7918	X	04
7919	V	04
791A	0r	4

Row Staua - Table (7AD7H - 7AE6H)

At the end of the K01M11Unification area is a 16-byte table, which is required by the ScreenEditor for row management. It contains a 1-byte entry for each of the 16 screens • ait the following content:

B80 - the row is IIIII
a single line of 32 characters.

81 - this line is the first of a double line of 64 characters

II - this line is the second of a double line of 64 characters.

Prograa - Table {Progra Status Table = PST>

A BASIC programme **entered or** read from the cassette **or** diskette is stored **in the programme table with** all its instruction lines. This **nor111connects** directly to the communication area.

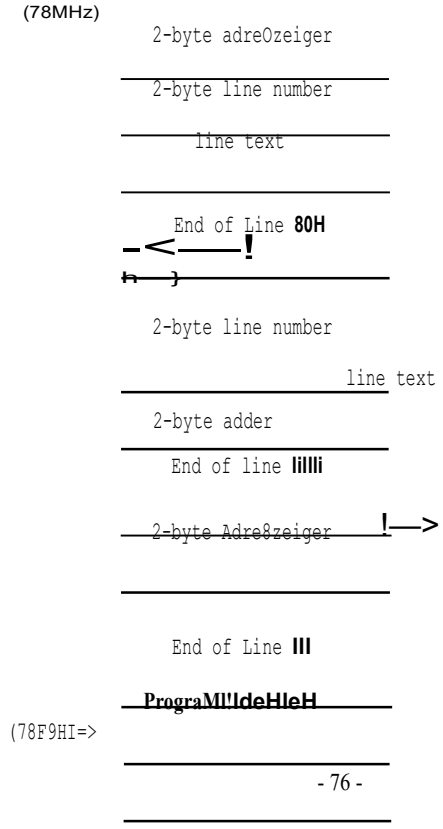
Since **the programme table** is in **the RAN** free-space zone and its **location** can change **if necessary, ia** is **in the** 7BMHs

a pointer to the table's initial address, and at 7BF9H, a pointer to the table's leading+t.

The input routine k011pri11 replaces the BASIC keywords with TOKEN and transfers them to the programme table. The individual Pr09ram11lines are stored ascending by LineNumberM!rn, regardless of the order in which they are entered.

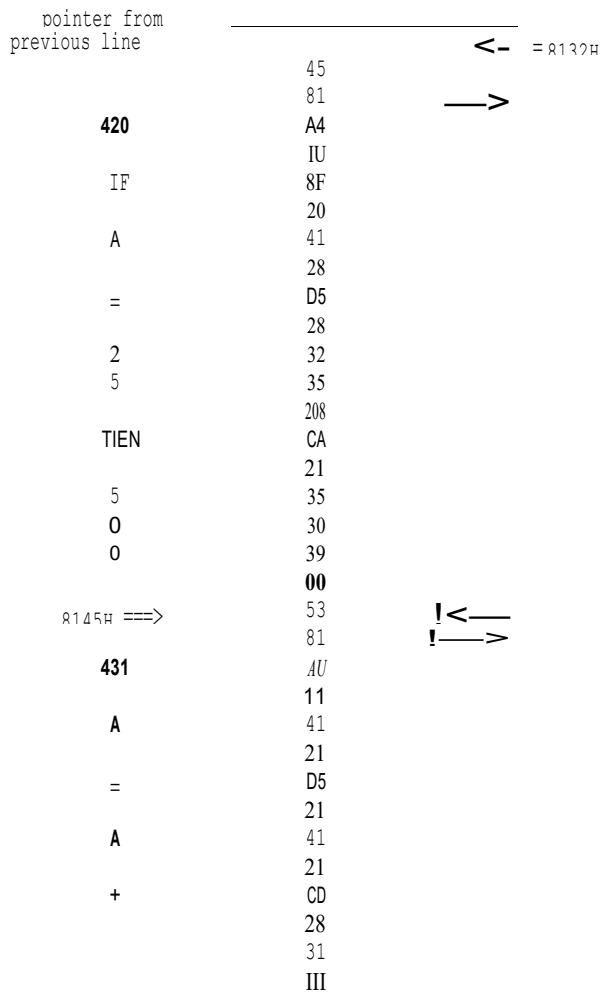
Each programme le starts 11it an N adreOzeiger to the beginning of the next line. Then the line number in 2-byte integer (ganzzahliget1) format. Behind the line tab is the actual line text, which is ended with a 'null' byte (0111H). This 'null' byte is also known as the 'End of Status' flag or EOS.

The Progra11111end 111is marked by two more 'null' bytes after the last programme line.



The contents of the PrograM table should be displayed using a simple example v0n two programme lines.

```
420 IF A = 25 TIEN 5111
438 A=A+1
```

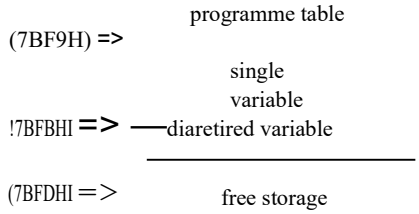


The variables - table

This table contains all the variables defined and assigned in a BASIC programme11111rden. The table is divided into two sections. Section 1 contains all simple variables, and Section 2 of i11 lists all the diversified variables, i.e. matrices.

Three adre$pointers in communication area provide information about the location of the variable table i・ RAN area.

7BF9H - initial address of 1. Section (single variable)
7BFBH - initial address of 2nd Section (matrices)
7BFIIH - End address of table + 1 (= start of free memory)



Regardless of **which** section defines a variable, the first three bytes of each entry have the same format.
Byte 1 contains the type code of the variable (82, @3, 04 or **8),** which corresponds to the length of the value part. Bytes 2 and 3 contain the variable name in **the** order 'second letter/first letter'.

I - Section (single variable) follows the value in the sequence 'LSB ・・・・・ ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ If text variables (strings) are 'Length' and 'Address' on the string ia String range or in the Progran table.

Entries i・ Section 2 have another header entry after the 3-byte header, which contains information about the size of the matrix.

The variables are entered in the variable labels Je during the propagation run and z111ar will be helpful when a value assignment is made or a latrix **is** created by DIN statement.

Within the sections, newly occurring variables are simply appended, there is no alphabetical collation. Since entries are only then pre-recorded 111erden, llll!nn a variable occurs during the programme run, it can happen that the entire 2. section in memory **must** be moved to space for a simple variable **i** 1. section.

Example: I • Progra111111, the variables A, B and C(5) have already been defined and entered in the variable table.

If a variable D follows, then the tatrix C(5) must be moved in
the second section of the table,
50 daf D • insert end of first section llll!rden.

Altatrizes **are** stored in i • Section 2 50, that$ edits the indexes from left to right llll!rden

Example: DIN E{2,3) **would be** saved **as** follows:

E!ll,ll
E(1,0)
E(2,0)

E(8.3)
E1.3)
E23)

The position of each element in the latrix can be determined with the following Frel

(Example of a three-diaensional latrix)

INDEX = !DRI **t** Gl'll + DNI l **t** 61..1 + 11..1

GNI = Average Index Limit + 1 Q..l =
Left Index Limit + 1
DRI = defined right index
DNI = defined DLI = defined left
index

Example: A l'latrix 11111rde ait DI" A(5,4,41 defined.

The item A3,1,2). This results in:

GI = 5, GI= 6

DRI = 2, DNI = 1, DLI = 3

INEX = (2 +5 + 11 t 6 + 3

= 69

AI3,1,2) is the 69. Element of the matrix.

The routine for calculating the indices can be found in i1 ROl1 at address 2795H.

Examples of different entries in the variable table

1. Simple Variable

Value	82	
length	8111	
Nar,e	43	C% = 11B
Value	64	
	8111	
Value	04	
length	110	
Name	44	D=-4
	8111	
Value	8111	
	80	
	81	
Value	83	
length	8111	
Nar,e	41	
	83	At= • XVZ
Length	nn	
Address		

2. One-di111ensional natrix

DIM A (20)

value length	04	
Matrix ball!	**00** 41	
Length of	n	= Distance to n. matrix
Natrix	n	
n/a indices	IH	
Max. Index+!	15 0o	
A(111)	LSB NSB SR EXP	
A(1)	LSB NSB #SR EXP	
A20)	LSB NSB NS EXP	
next	04	
Natrix	1111	

3. Three-diaensionary matrix

DIN A (4.5.9)

value length	04	
sodium name	**00** 41	
Length of matrix	n n	= Distance to next 11atrix
n/a indices	03	
11ax.Value	**IIA**	
right.Index+!	%	
Nax.Value	86	
11itl.Index+1	**0o**	
11ax.Value	05	
left. Index +1	**00**	
A(0,0,0)	LSB NSB 11SB EXP	3014 Byte Entries
A1.8.0)		
A4.5.9)	LSB NSB 11SB EXP	

next
Natrix

1. The Mutzumg of the BASIC stack

Before the string area, the stack area of the BASIC interpreter is located at the end of the memory available for BASIC.

Ia Usually the communication area serves the BASIC interpreter as a tool for caching work values and addresses. This is not a good thing, however, since some routines might call themselves internally (recursion) and overwrite the previously saved intermediate results for the file. An indexed table would also help, but the BASIC interpreter uses the stack to do so.

In addition to the normal caching of register content and return addresses, the BASIC interpreter stack is used primarily for three special functions:

- FOR/NEXT Loops
- 60S\JB - Views
- Expression Analysis

stack in F9/NEII loop

If a FOR statement occurs, all required variable addresses are packed into a FOR block on the stack. If a NEXT statement occurs, the FOR block is searched with the corresponding run variable on the stack, which is located at 1936H.

The stack is searched from back to front. If no matching FOR block is found, the error message is

NEIT VITHUT FOR

.

Form of a FOR block:

B1	FOR - TOKEN
LSB	Address del'
IISB	Run Variables
LSB	
	survey value
IISB	
LSB	
	final value
IISB	
LSB	Lennuer der
IISB	FOR instruction
LSB	Address of first
IISB	loop statement

Stack Usage at a 6 – Statement

When a GOSUB statement **is** issued, a 7-byte block **is** written to the stack, which is obtained and evaluated by a subsequent RETURN.

Form of a GOSUB block:

91	GOSUB-TOKEN
LSB 115B	Line *number of the* GOSUB statement
LSB IIS B	Address of the GOSUB statement in the Progran table

Expression analysis breaks expressions into their individual elements and connects them according to the precedence of operators within the expression.

Each expression is searched and the highest value operation is performed first. The resulting intermediate result will be cached and the next higher operation ia Expression er111will be cached and executed. This will continue until the expression is fully resolved.

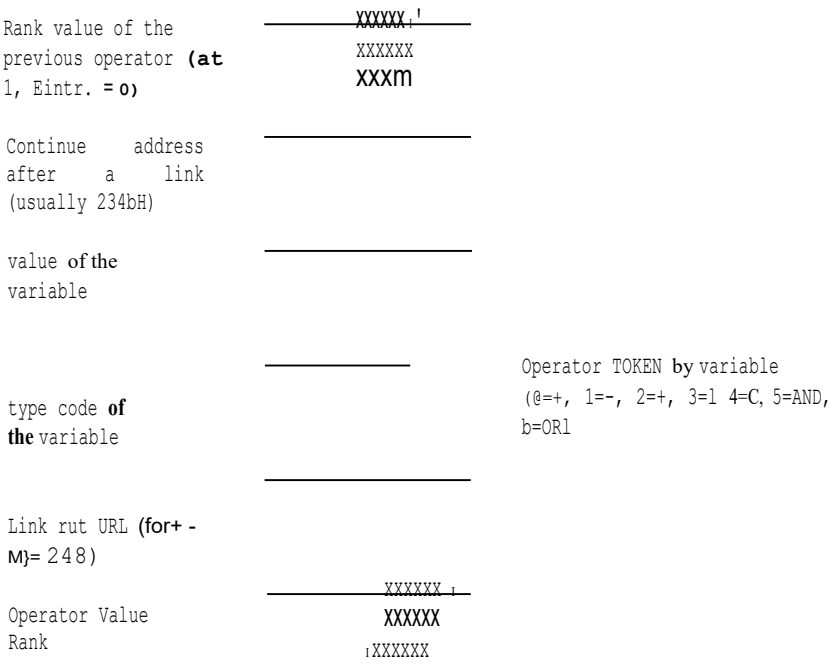
An expression 11 is searched from left to right. The search is interrupted if an operator or the end of the expression is found. The variable on the left of the found operator (referred to as current variable>, zusa1111111en with the operator {an aristocratic link synbol +, -, %, /, [) are called the set and either

- if the rank of the operator was greater than the previous one, written as a sentence to the stack; or

- if the rank of the operator was equal or lower than the previous one, the variable associates with the previous set of Stack.
  The previous sentence is removed from the stack and the result of the link is considered a new 'current variable'.

This will be repeated until a new sentence created in this way is pushed onto the stack, or no more values will be placed on the stack to link. In such a case, the expression was completely dissolved.

The variable/operator sentences are written to the stack in the following format:



It is relatively easy to check whether or not a link should take place. The rank value of the operator in the last sentence is *the* last entry on **the** stack. It is checked whether the *new* operator is equal or smaller in rank value. If not, the new operator and the current variable will be written to the stack as a new sentence. If so, a link will be performed.

In case of a link, the last sentence is fetched completely voii Stack and the link rutine specified there (usually at 24%}) is jumped. There the previous variable of Stack **is** linked to the current variable according to **the** previous operator. The result will be the new current variable, which forms a new set **with the** current operator.

-

The link is then jumped back to where it is checked again whether the new sentence in rank value is less than or equal to another sentence in the Stark.

If the stack no longer has a set, or if the existing one has a lower rank-value than the current one, the newly formed sentence is written on the stack. Otherwise, a link will occur again.

The end of a statement or the occurrence of a non-arithmetic TOKEN triggers a cross between the two.

The following example should represent such an expression analysis in incremental steps:

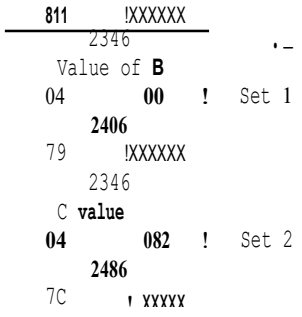
Expressio n ' A=B t C t D/E c 5

The search begins with the first character on the right of the equality sign and ends with a11 '+' character.

'1' and '+' are written to the _Stack as the first sentence, because there was no comparison set there, or the one stored there at the initialisation reflects a lower rank value.

The search will resume and stop again at ' '*'. The variable/*operator* - set '; + is written to the stack as a second sentence, because the rank value of the '#' operator is greater than the '+' on the stack.

The stack now looks like this:

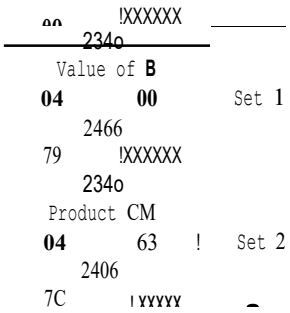


A new break occurs with **delll** character '/'. Now a link has to be performed, because the rank values of'*' (on **dell** Stack) and'/' (the new operator) are the same.

Sentence 2 **is** read **by** Stack and is branched to the link routine at 2406/i. There the link between 'C *' and the current variable **'D' is** performed, i.e., '' **is** multiplied with 'D 111. This results in a new variable, the product of 'C * D'.

After the l'lultiplication, the Progra11111 is continued at 2346H and the new variable/operator set **'C /' in rank-lert is** compared **with** the sentence 1 on **the** stack. Because the rank value of '/' is greater than that of the first sentence('+'), lll'C*D /' is stacked as a second sentence.

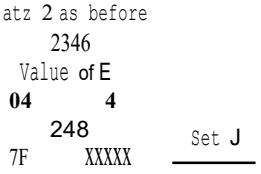
Now the stack has:



The parsing of the expression continues behind **delll** '/' and stops again when the character '[' is reached. The current variable/operator set 'E[' 111is brought to the stack **as a** set J, because the rank value of '[' is greater than the previous '/'.

This results in the following image on

dell Stack: ! Sentence 1 u.atz 2 as before



If the search is continued, the end of the expression is found after the number '5'. As I said before, reaching the end of the expression triggers a link.

Set 3 is fetched VOii Stack and 'E [5' is calculated. The result is the new current variable,

Since the current operator i1111er is still reaching the end of the printout, IIIII!rden also fetched sentence 2 and sentence 1 from Stack and the operations

C * D / E t 5

and last

R+CM+D / Et 5

.

Then the Progra at 234! is not another element of the expression 11ehr on dell Stack and it is jumped back to the calling routine.

The expression was de evaluated, the result is located in i ' workspace 1 of the CONUnikationsbereich (X-Register) and from there the variable 'A ZU9wiesen,

The BASIC interpreter supports 16 arithmetic functions, of which the following 7 liltheatic functions.

Sine	(sink	**(a)**
exponential function	**(e)**	(bl
Arkus Tangens	(arctanl	!cl
natural logarithm	Uni	(dl
Cosinus	(cosl	(a
root function	(x)	**(t)**
tangent	(tan)	(g)

The functions e-g can **be** expressed by the functions a-d**.**

1. co = si ('f + f!: **i radian**
   2

(2) tan 'f = sin 'f **i radian**
   cos P

(3) 7V7 MV2 1 **e =** Euler number (2.71 .....)

If (3) is generalised:

(4) **=** 9 lnx

For internal calculation, **the** BASIC interpreter uses arithmetic approximations of the functions a-d. All other functions **lill!rden ait** calculated **to** help the approximations and the functional additions111111 (1)-(4).

According to the laws of the power series development, the sinus can be expressed in a number **who the.**

$$(5)\ \sin/o = \sum_{n=1}^{\infty} \frac{(-1)^{n} f^{2n+1}}{(2n+1)}$$

P 12 development point o = 8
ill logenaaf
lf(xll amount f(xl

$$= f_1 + \frac{LT^3}{1+2+3} - \frac{\quad}{12+3M45}$$

The BASIC interpreter approaches the sinus up to 5. Song. The angle is a multiple of the arc (tl).

$$(5.1\ P = 2+ \quad ; \text{ Pis Bogena}$$

$$(5.2)\ \sin(2rt = 277t - h+3 - \{l - 9 \quad \text{Three!} \quad \text{Five!} \quad 7! \quad 9!$$

If x is given in degrees, t is also calculated.

$$(5.J)\ t = \frac{2}{f} \cdot \frac{399}{x} \quad \text{X in degrees, } f \text{ ia Bogenaaf, } 1\ t\ 1,\ 1/4$$

For **marking at angles between I** and J6II **degrees** (1 *f' l* < 211' >t is calculated:

$$(5.4)\ t = \frac{x}{J6IO} \quad ; 0^{o} < x < 90^{o}$$

$$(5.5)\ t = \frac{181'1}{J6IO} - \frac{x}{3} \quad ; 90^{o} < x < 270^{o}$$

$$(5.6)\ t = \frac{x}{J6IIO} \cdot \frac{38}{'311/P} \quad \$2789\ x(309$$

At larger angles, the number *of* whole multiples *of* a circular arc shall be deducted and then ·The above shall be done.

The elements *of the* individual rows are accurate to four decimal places.

The **naxial** error *of* approximation is <0.HM35 (for !t! <1/4). This allows the **health outcomes** to be specified in five digits.

exponential function

The BASIC interpreter calculates the exponential function *e'* for all *values* -

88 (x(**88).**

The functional approximation has two elements. One element is an integer exponent to **base** 2, the **second** a series development in 8 links.

) **e'** = 2x los2 e

6.11 $e^x = e^{-t} (2^{[x \log_2 e]+1})$;ftx) = Stairs function largest integer in f(xl, known as Integer.

Ot{In 2

(6.2) $e^x = T e^{-t}$ r2, go, el+1

Here *he* describes the difference between e and detl next larger integer multiples of 109, *e* as an exponent.

(.3) t=-x+lxlo elln2 + 1n2

6.4) x= 1ne'= 1n (et2[x los, *el*1)

= -t + In 2 (each $\log_2 e$ ll +1 › ;8t({In2

The integer potency to base 2 can be directly bestiMt rden (binary computing systems).

The power series for the second element is general:

$$(6.5)\qquad et = 1 + \sum_{n=II}^{oo} \frac{(-+n}{(n+1)}$$

$$t2$$

$$= 1 - x+$$

The results found from (6.5) and (6,2) accurately indicate 5 decimal places.

Arctic days

The approximation of the Inversen Tangens is based on the series feeding up to 9, link.

$$(7.1)\qquad \arctan x = \sum_{n=II}^{OO} \frac{2n+1}{(2n+1)}$$

$$= x - \frac{\#}{3} + \frac{1.5}{5} + 7 \quad \frac{}{7}$$

In the case of negative input values, the amount of x is calculated and the result is reversed. For values x > 1, the approximation for 1/x is calculated.
The result is:

$$(7.2)\qquad \arctan x = \frac{77-}{2} - \arctan \frac{}{x} \qquad lxl > 1$$

For values !x! < 1 results directly from the series development. The coefficients 5-7 are corrected so that the aaxial error is 1.126 (i • logenaajl).

$$(7.3)\qquad \arctan x = x - 0.\ 33331\ " + 8.19993\ "° - 0.1420889\ 7 +$$
$$0.\ 10563\ 7 - 0.075289\ 4! + \mathbf{0.042999}\ 43$$
$$0.01616157\ 415 + 0.00286623\ 17$$

The approximation of the natural **Logarith1MJ is** calculated from three parts of the corresponding series.

$$(8.1)\qquad \ln\frac{1+a}{1-a} = \sum_{n=0}^{\infty} \frac{a}{2n+1} \qquad \text{for all } <1$$

$$(8.2)\qquad x = \frac{1+a}{1-a} \qquad \text{for all } <1$$

Dar from it results'

$$(8.3)\qquad \ln x = ?\left[\frac{x-1}{x+1} + \frac{1}{3}\left(\frac{x-1}{x+1}\right)^{3} + \frac{1}{5}\left(\frac{x-1}{x+1}\right)^{5} + \cdots\right]$$

This series converges for values $x(1$. Therefore, nuss$ must be transferred by scaling x into these areas,

$$(8.4)\qquad x = y2^{n} \qquad 1/2 \sim y < 1,\ n = 1,1,2 \cdots$$

The scaling factor is the next higher power of two 'it integer!lll exponents.

$$(8.5)\qquad \ln x = \ln(y2^{n}) = \ln y + n \ln 2$$

$$(8.6)\qquad \ln x = \left(\frac{\text{ed } y}{\text{ed } 2} + n\right)\ln 2$$

$$\ln\left(\frac{y}{\text{ed } 2}\right) \qquad \ln < \ln 2$$

$$= \ln 2 \left(\frac{\text{ed } 2}{\text{ed } 2} + n\right) \qquad \ln 2$$

The constants of this formula are specified internally:

$$\ln 2 = 8.07870892$$

$$\ln(\ln 2) - \cdot \text{ II } \ln 2$$

$$\text{!B. 7)} \qquad A = \frac{\ln(-)\ \text{in } 2}{\ln 2}$$

The scaled value 'A' can be calculated using *the* approximation (8.8).

$$\text{(8.8)} \qquad = -\left(\frac{2}{\text{ed } 2\,y} + \frac{\frac{y}{--1}}{\text{ed } 2}\left(\frac{1}{3} + \frac{1}{5}\right)\right)$$

$$\text{(8.9)} \qquad \ln x = 0.707092\ (A - 0.5 + n)$$

The accuracy is given in four digits. For *values* of x in *the* proximity or *door* very large *values* cannot achieve this accuracy,

The BASIC interpreter of the LASER computers 110, 210, 31@ and the VZ200 consists of a large number of closed routines that are called to execute besti11111ter functions by the execution control. Many of these routines can also be used by machine programmes and thus simplify programme creation.

This chapter describes a number of these routines and shows their integration into l'laschinenprogra1111111e using call examples.

It is important to know exactly what the input and output conditions are when using **the** individual routines. Where are the input parameters to be provided, in which data format do they need to be provided and **where** and how is the result provided? These are just a few *of the* questions that are coming up. It is also important to know which registers will be changed by the routine that is running so that you can secure them in good time.

In this context, 'ltll!Tlhang should also be reminded again, since0 cannot change the register pair **IY** when connecting to a floppy system,

Eiw/Ays, Jabrjlgutinen

This section describes routines that deal with the 111it of the interpreter's communication ·his fullfield.

How can I read characters from the keyboard, display them on **dell** Bildschir111 or print them on a printer? How can you use the cassette interface from machine programmes or output sounds to **the** small built-in speaker?

Of course, you can **evaluate** the keyboard matrix yourself. But there are also ROM routines available to do this work.

CALL 2BH — Evaluate Keyboard

This routine evaluates the keyboard matrix once and passes the result. Holding down the key, the ASCII code is determined and entered in the A register.

It immediately jumps back to the calling routine, regardless of whether a key is pressed or not. If you want to read lll!your characters directly in succession, you have to ensure *a* debouncing yourself.

Changed is the DE register pair and the A register, in which the result **is** transmitted.

Example:

```
                LD      C, 688H     Possibility of
                CALL    6111H       debouncing
                PUSH    EN          ;Save EN
                CALL    2BH         ;Read keyboard
                OR      A           Was the key pressed?
                JP      Z,NO        {=} to My Branch
YES             POP     EN          ves branch
```

The A-Register contains the ASCII code of the pressed key. If no key is pressed, the A-tab is empty **{H), the** above example invokes the read routine and, depending on whether *a* key is pressed *or* not, either branches to the NO routine or continues the Progra1111 during *the* YES routine.

'CALL 2BH' offers it one time when you want to see if someone rang the door during the programme run, i.e. a key **was** pressed.

CALL 49H — Waiting for keyboard input

Here you will be checked0llten, whether a key was pressed or not, you will only get the control back, IIIE!nn someone actually pressed a button. Otherwise the routine corresponds to the call 'CALL 2BH', which is also used internally,

Example — To a given question the user should answer 1it 'J' = Yes or 'N = No,
The PrograM waits for one of these buttons to branch to the Yes or No routine.

```
LES     PU5H    EN          Secure ;DE
        CALL    4%          wait for key press
        POP     EN          ;Restore DE
        CP      'J'         'J' key pressed?
        JR      Z,YES       {=} to JA Branch
        CP      'N'         Was it the 'N' key?
        JR      NZ, LES     Not even, wait

NO                          no branch

YES                         ;YES branch
```

The characters read in 'CALL 21!H' or 'CALL 49} will not be on dell! Display • You may need to do this yourself in one *of the* following output routines.

CALL 3E3H — Import a Line

KD11more advanced than the two previous calls, you will be served by the routine at JEJH.

There a full line is read from the keyboard and displayed on dell screen. The read-in line will then be provided in the input/output buffer of the Konun ikat ion section for further processing.

The display on **dell** Sieba starts at the current cursor position and the cursor will automatically flash out at aut011atic.

An input line can be up to 6 characters long and must be completed using the <RETURN>key or <CTRL-BREAK> key.

Since this routine is used, among other things, for reading BASIC programmes, only the characters valid in it are allowed (letters, digits, special characters>). If you also want to read block graphic characters or inverted characters, 1You must include them in quotation marks when you type them, If you enter incorrect characters, the text 'SYNTAX ERROR' appears and the input can be repeated **by the operator.**

This routine requires that the interrupts are enabled (EI= enable interrupts).

After entering, the read-in text in the input/output buffer is from address 79E8H. The register pair **II.** points to the byte (79E7Hl. The text end is marked with a character of **111H**.

Using **the** CarryFlags, you can determine if the text **has been** completed with ({RETURN} **of the**<CTRL-BREAK>).

Carry =	1it <RETURN> completed
Carry = 1	1it <CTRL-BREAK> complete

All registers **are** changed by *the* routine.

Example: A line of text should be read from *the* keyboard and transferred to the 'TEXT' field, register B should then display the text length.

```
        EI              ;Enable Interrupts
LES     CALL    3EJH    Read Line
        JR      C,LES   ;BREAK - one more time
        INC     th      ;II. to buffer start
        LD      EN,TEXT ;DE= Text Area
        LD      B,0     ;Character count= 0
TRF     LD      A, H)   ;1 Buffer characters
        OR      A       End of text?
        JR      Z, FINI yes" finished
```

```
        LD      (EN),A      Draw in text
        INC     ed          area ;Buffer address+
        INC     EN          1 ;Text range+ 1
        INC     B           Character counter + 1
        JR      TRF         next character
FINI
TEXT    DEFS    64          ;Text Range
```

For FINI, the **entered** line is in the 'TEXT' tab **B** and contains the text length.

Show characters on **the screen**

Three routines allow you to display text on **the** screen. With 3Al you can output a single character, with 28A7H and 2B75H you can output a whole string.

The text viewer is located at 70111+-71FFH. You can also write **something** directly in this area, but should note the Bi ldschira synchronisation **i** but also this l'lölichkeit t is shown in an example.

CALL 33AH — Show a character

The i1 ,\ tab provided character **is** displayed at **the** cursor position on **the screen**. **The** cursor **is** preceded **by a** character.

I A-Registers shall **use the** normal ASCII codes**,** not the internal **Display** LASER **codes**.

In addition to the alpha-numeric characters, control characters **of the** ASCII code table are recognised and the corresponding screen1 functions executed.

08H o. 18H		Cursor one place to the left
09H	o.	Cursor one place to the right
19H0A		Cursor one line to the bottom
0DH		To the beginning of the next line Insert
1SH		a space (INSERT)
1BH		Move cursor one line up
I		Cursor to the upper left corner
tDH		To the beginning of the line.
1FH		Delete Image
7FH		Delete characters at cursor position (RUBOUT)

Register contents are not changed, they **are** secured by the routine during tension and restored **again** when **jumping back**,

Example: At **the** cursor position the letter 'A' should **be** displayed.

```
LD A, 'A' CALL    Load ;and
3JAH              Output
                  Characters
```

CALL 2BA7H
or
CALL 2B75H — Output a Line

If you have more than one character to output, you could easily cross the call at 3JAH in a loop lll! The routines at 28A7H and 2B75H do this work for you and provide a full text **with it.** single CALL call to the screen.

The text start address **must** be provided **in** H register pair, the text end must be marked with it, at 28A7H 8DH is also recognised as text end.

The routine at 28A7H uses the string cache in Kaun i cation range and the BASIC ring area at the end of the memory for text preparation. Therefore, you should only use this routine if the configuration and tri ngbereich of your machine programme are properly managed.

The routine at 2B75H performs a direct output and is not subject to the above. Restrictions. l'lit this routine will offer you another comfort. By simply switching a byte you can update the output from the screen • to a connected printer or even to the cassette recorder pages. This is the byte 789CH in the K0111111.1nication range.

00H = Screen 1H =
Printer B88H=
Cassette

But do not forget to put this byte back on the screen output. Otherwise, later system output may migrate to the other device.

Example: The screen should be deleted and then *the* text 'l'IEIN LASER IST FANTASTIC' should be displayed.

```
        LD      HL,         ;Load text
        TEXT                address ;Output
        CALL 2B75H          text


TEXT    DEFB 1FH            ;Delete Screen
        DEFl'I 'L'IEIN LASER IS FANTASTIC' DEFB
        1 ;Text ID
```

CALL 1CH — Delete the screen

The screen is clean with a CALL 1C9H. Internally, it does not seem to output the control characters 1DH (CUrsor to the beginning of the image) and 1FH (delete image) with two consecutive CALL 33AH.

Example

```
        CALL 1C9H           ;Clear Screen
```

Direct output to **the brain**

If you want to write directly to the screen memory (7000H-71FFH), you must not use **the** normal ASCII character set. The IIIUß character to be output in the LASER internal encryption provided 111erden, but also block graphic characters of all colours can be output directly (see Chapter 5). The process is also suitable for high resolution graphics (70011H-77FFH). 11!!Be careful, because each 1\yte contains the information for four pixels.

However, such output should be synchronised with **the** image generator, otherwise the image quality will suffer (in case of frequent output, the image will be disturbed by crossstrokes).

One possibility *of* synchronisation is ***the*** RAl'I extension output *of the* lnterrupt-5ervice routine at 787DH. There you can use a jump 'C3 xx x>:' to your own routine. After writing your text in this Rootine, return to the normal interrupt service routine with a simple RETURN {€9H).

Another option, if the interrupt (DI) is switched off, is to query the bit 7 i • 6800H-6FFFH input/output range (see Chapter **4),** this bit is directly connected to it **dea** vertical sync signal of the image generator.

Example **Type** beginning *of* 3. The letter **'A'** should appear *in the* line *of the* screen**.**

```
        DI              ;Disable Interrupts
        LD      H, 7040#  3. Address Row
        LD      B,'A'     ;Load character to output
WAIT    LD      A,(6800%) ;Check Vertical Sync
        OR      M
        JP      P,TYPES   if bit 7 = 8, wait
        LD      (HL),M    ;Print Characters
```

From a single 11machine prograH, RON routines can use 111erden to output characters to an attached printer. One of these possibilities 1111.1rde already discussed at the 11it **dell Clil 2175H and** *the* roll-*up* of the flag at 789H. This allows entire text lines to **be** transferred to the printer. A **further** possibility is described below:

On a printer the mark "Seikosha 6Pll0[1] (see chapter 8) can also be printed *of* block graphics and inverted text characters.

The single character 111 output routine automatically starts to "finish" the printer. However, you also have the possibility to ask the printer status.

CALL 3BH — Print a character

The character passed in the A-Register (ASCII code) is written to a connected printer.

Ia Device Control Block **at** 7825H-782H is automatically executed a line counter, which is reset to **6** lines. These **6** lines per page are a set of values when the caching area is initialised and are located at address 7828H (number of rows/page+ 1). Lines/page corresponds to 11 inches and thus dell a11American sheet format, The German DIN A4 format has about 12 inches, i.e., if you want to use the sheet feed control, you should change this value to 72 lines/page (+1 =494).

In addition to the standard ASCII text characters, control characters can also **be** transferred to the printer, e.g. Escap sequences for font selection (see printer manual)

The following control characters **are** already recognised and executed by the printer driver:

1111H	The printer status **is** obtained and returned in bit **8 of the** A register. With **= 8**- *the* printer is ready to print Bit = 1- the printer is not ready The Zero flag is set accordingly.

'MBH	Absolute sheet feed. The printer is placed at the beginning of the next page.
OH	Conditional sheet feed. **The** printer will only be placed at the beginning of the next page if *it is* not **already** at **the** top of a page (line counter at 7829H = 01).
0DH	Carriage Return (CR) A carriage return character (0DHJ and a line feed character (0AH) **is** returned, Caution: You should set your printer **to** not automatically feed **lines** after a carriage returns (see the printer's manual). Otherwise, a blank line will be inserted.
	Line feed. Is internally converted to a @DH before execution,

Determine Printer Status

CALL SC4H

By calling this routine, the printer status can be directly queried !only the 'BUY' line is monitored).

The status is passed in the @ bit of the A register, and the ZERO flag is set accordingly.

Bit 0 = **0** IZ flag = 1) - The printer is ready.

Bit @= 1 {Z-Flag = **Q)** - The printer is not ready

Example

The text 'TEST A PRINTER OUTPUT' should be printed on **the** printer.

If the printer is not ready to print, the text 'PRINTER NOT READY' should instead appear on the screen.

```
         CALL    5C4H        Verifv Printer Status
         JR      NZ, ERROR   ;not ready
         LD      tL,DRTEXT   ;Address text
LOOP     LD      A,(H)       ;Load text character
         OR      4           ;text?
         JR      Z,DONE      Yeah, done
         CALL    3BH         Print Characters
         INC     th          Address next character
         JR      LOOP        and spend
DONE

ERROR    LD      tL, FETEXT  ;Address error text

         CALL    2B75H       ;Display on Screen

TEXT     DEFl'I  'PRINTER OUTPUT TEST'

         DEFB    0
FETEXT   DEFl'I  'PRINTER NOT READY'
         DEFB    0
```

The Cassettes - Input-'Output

The KOM1Jnikation ait a connected cassette recorder is carried out via the input/output range 6800H-6FFFH (see Chapter 4). bit is used to read the information from recorder" about bit ! and 2 is output to the cassette recorder. In contrast to the screen and printer, the transmission is serial, i.e. for a character to write or read (= 1 byte), 8 bits must be written or read in a row.

There are a number of routines available that you can use when editing the cassette interface. The default for11at of a recording can be found in Chapter 8, Section The cassette driver.

Since the bit recording is very time-critical, it is important to disable the interrupts !DI> before editing !Read or write) in any case, otherwise you do not transfer any useful information.

Writing on the cassette

CALL 3511H — Write a byte on the cassette

In this routine, a byte **is** output bitterly to **the** cassette recorder, the byte to be output shall be provided **in the** A register,

CALL 355BH — Write File Header to Diskette

This will output a complete file header to the cassette recorder, which consists of the sync bytes (255 y Bai) 1 dl!fi Before spann (5 FEH>, detl file identifier (Ffli = BASIC, FlH = binary, F2H = data file) and your Names (Max. 15 characters).

The file name shall be provided in a separate field, enclosed in quotation marks, **H *u*** included in the initial address of this field. The file identifier shall be passed to the C-Register.

If the carry bit is set **when** bouncing back, **the** recording process was interrupted by the <CTRI... BREAK> keys,

Example: **The** memory range from **801118H** to SFFFH should be output as a binary file 11it **del!!** Name 'TEST' on **the** cassette recorder 111. A test sample! shall be written behind the data for a later load check,

```
DI              ;Disable Interrupts
LD    C, 8FIH   ;Binary file expert
LD    LT,NAI'E  Address File Names
CALL  355       ;Output File Header
JP    C,BREAK   ;BREAK key pressed
LD    BC, 400   a small break in between
CALL  611H
CALL  JAESH     ;BREAK key pressed?
JP    C,BREAK   Yes!
LD    IX, 782: H  Addressing Checkstring Bytes
LD    H, 8000H  ;Load Startup Address
LD    AL        LSB Start Address on cassette
CALL  3511H     ;Write
LD    (11),A    and in Pruefsunae
```

```
        XOR     A           #SB Checksunae =
        LD      (I+1),A
        LD      A+H         ;NSB of the start address
        CALL    3511H       ;Run on cassette
        CALL    388EH       ;and add to probe
        EX      EN, Hl      ;Start address in DE
        LD      Hl, SFFFH   ;Load End Address
        INC     Hl          ;+ 1 for recording
        LD      sL          ;LSB End Address
        CALL    3511H       to print on cassette
        CALL    388EH       ;and add to Checksum111111e
        LD      A,H         ; NSB End Address
        CALL    3511H       to print on cassette
        CALL    388EH       ;and to test UIIIIIE! add
LOOP    CALL    JAEBH       ;BREAK key pressed?
        JP      C,BREAK     Yes!
        LD      A, DE)      ;Load bytes of memory area
        INC     EN          ; Memory Address + 1
        CALL    3511H       ;Output Byte to Cassette
        CALL    388EH       ;and add to checksum
        RST     18H         End reached?
        JR      NZ,LOOP     ;no, next byte
        LD      A, IX)      {LS} of checksum
        CALL    3511H       ;Run on cassette
        LD      A, IX+1     ;/SR of PrGfsune
        CALL    3511H       ;Run on cassette

        EI                  Reset Interrupts

BREAK


NA}E    DEF11   ·           ;File Name
        TEST'
```

The above example invoked some routines that are not yet described.

CALL 3AEBH — Remove BREAK key

This rootine ' checks to see if the <CTRL> and <BREAK> keys are pressed at the same time (BREAK function), If this is *the* case, the CARRY bit is set.

CALL 3BBEH — Create Pr. Form

This routine is used by the cassette recording routine and *the* read routine, and the checksum of a record is determined. The two bytes 7823H and 7824H are available in the cognamentation area, which must be addressed and initialised with the register pair IX.

RST lBH — Comparison HL to DE

This logically compares the register pair H to the register pair DE. The CARRY and ZERO flags are set according to the result of the comparison.

When handling the RESTART procedures ' this routine is described in detail.

Read from the cassette

CALL 3775H — Read a byte from cassette

This reads a single byte from *the* cassette and makes it available in the A register. The BC, DE and II register pairs. shall remain unchanged.

For read errors, the CARRY bit is set.

Before reading a byte to the reader routine synchronised to a valid record.

When reading older bytes, the baud must be kept rhythm.

CALL 35E1H — Find file on cassette

This routine detects the beginning of a file on the cassette and synchronises the reader routine to the recording.

The name of the file to be searched is to be stored in communication area from address 7AB2H and completed with 00H, for this purpose also the routine at 358CH can be used, the starting address of the name field in HL is to be handed over.

If a file of the specified name is found on the cassette, the file identifier is transferred to field 7AB2H and can be checked there.

During the search process, messages about the search status are displayed in the last screen line:

WAITING - sync bytes not found yet.

FOND Y: File Name

A file with the specified names was found.
'I' = File Identifier
T = Text file (e.g., BASIC programme)
B = Binary file (e.g. Machine Programmes) D
= Data File

If the specified number does not correspond to the file you are looking for, the search will continue aut011atically and more recent FOOND messages may appear in a row.

The o.a. Messages can be suppressed, in which i byte 74CH of the costing area is entered a value unequal.

If the BREAK key is pressed (CTRL-BREAK) during the search, then the calling programme will skip back to the BASIC+taupt loop not 2LIII.

Example: The amount of memory recorded in the previous example should be re-read. The checksum shall be determined and verified at the end of the recording. The message output should be suppressed.

```
         DI                    Disable {Interrupt
         LD       + 1          Suppress H1Message Output
         LD       (784CH1,A
         LD       HL,NOSE      {File Names in Communication
         CALL     35BCH        Apply Range
SEARCH   CALL     35E7H        ;Find file on cassette
         LD       A, (7AD2M)   Check {Name found+ ID
         CP       11JF1H       ;Is it a binary file?
         JR       NZ, SEARCH   No, continue searching
         LD       IX, 7823H    ;Pr Gf summenbytes
         CALL     3868H        ;Read start and end address
         JP       C,FEHI.ER    ;Read error!
         OR       A            Delete Carry
         SBC      HL,DE        {Determine programme
         JP       C, ERROR     ;Start address> End address
         PUSH     HL           ;Broadcast length in BC
         POP      llC
LOOP     CALL     3775H        Read Cassette Cover
         JP       C, ERROR     ;Read error
         LD       (EN),A       to transfer to storage
         CALL     JBBEH        ;and add to checkpoints
         INC      EN           ;Memory Address+ 1
         DEC      BC           Length - 1
         LD       A+ C         = 0 ?
         OR       to
         JR       NZ,LOOP      No, continue reading
         CALL     3775H        Read LS} of Pr Gfsunne
         CP       (IX)         i= the calculated?
         JP       NZ,          ;no, Loading error
         CALL     3775H        Read 3/SB of checksum
         CP       I+1)
         JP       NZ, ERROR    no, load failure
         EI                    ;Enable interrupts,       done!

ERROR

NAI'IE   DEFI'I   '"TEST"'     {File Name
```

In example, two helper routines were used, which are to be cleared shortly after.

CALL 35BCH — Transfer Filename

The routine is to transfer a filename in the programme to the costing area from address 7A9DH.

The filename must be addressed with H before the call and included u$ in quotation marks.

CALL 38H — Load start and end address

If the correct file was found on the cassette, the start and end address of the memory area can be read from the cassette here.

• In case of return jump DE contains the start address and II. the end address +1 of a text or binary file.

The PrüfSUMe is initialised by this routine.

If the CARRY flag is set at • Backjump, a read error occurred.

Speaker - Output

The small built-in loudspeaker of the LASER computers and the VZ2IIII can also be accessed from PlaschinenprograNen.

This speaker is hard-wired, • as described in the beginning, with the bits @ and 5 of the input/output range 888H-FFF.

bit 5 bit 8

1
1

Speakers

These two bits know how to be complementary here. Toggling the bits at a certain frequency will produce a sound. However, you do not have to fear that you have to do this yourself out of a machine programme. Two ROii routines allow it to give enh1eder a single sound or an entire 11elodie ait einea call.

CALL 345CH

Make a single sound

When called, the register pair HL shall specify the pulse length (as shown in 2CFHl and BC in the register pair).

All registers will be changed.

To create a clean sound, the interrupts should be switched off,

Example:

```
DI              Disable interrupts
LD    H,0AM     Load pulse
LD    BC,       length ;Load duration
CALL  35CH      ;Emit sound
EI              ;Turn back on interrupts
```

There is a high short beep.

The higher the value in Hl. is, uiiso is getting darker the sound. The tone duration also depends on the pitch. h achieve the same tone duration with two different tones, to be set to a greater value for Gr RC at higher tones.

CALL 2BF5H

Playing an 11-hole

This routine allows you to play a full 11elodie ait einea call. The individual notes shall be specified as in a BASIC-SOUND command:

sound, length; Sound, length; Sound,Length3. - +...,

The melody is to be provided as ASCII string i11 PrograM and to be addressed with the HLRegisterpairn. Sound and length are separated by comma, several tones can be specified in a row separated by Se1likolon.

Example: Play a small 11elodie

```
        LD    HL,        ;telodie string address
        NELO             teldie play
        CALL 31 FSH

NEL0    DEFM  '16.2121.2521.2123.2523.2525.326.13'
        DEFN  '28,252,2525,2323,2323,2421,4°
```

Conversion lutins

data type transaction

A set of subroutines is used only to provide the numeric data in the correct type before processing, i.e. to transform it from one 111 data type to another 1 111.

The conversion routs in the list expect the value to be converted i Labour Register 1 of the Ka..mikationsbereich <X-Register) and the data type of this value ia TypFlag at address 78AFH.

The result will be made available again in workspace 1. The TypeFlag 78N% contains the number plate of the new data type after the landing.

CALL A7FH — FlieJk01111Number in Integer

The contents of workspace 1 are converted from a variable of simple or double precision to a number {Integer).

All registers will be changed.

There is no rounding.

Example: The value 2.88539 should be converted to v01II type "simple precision".

```
        LD      DE,VALUE        {Address Source Value
        LD      ll., 7921H      ;Address workspace 1
        LD      RC,             ; value length= 4
        LDIR                    Baseline Value in Workspace 1
        LD                      ;Type flag = simple precision
        LD      (78AFH),A
        CALL    8A7FH           ;Call conversion routine
        LD      H, (7921H)      ;Result from workspace 1
        LD      (DECORATES),    transfer to result field

AIERT   DEF! 45H                Baseline value 2.88539
        DEFB 8AAH               as flow
        DEFB J8H                coefficient ;simple
        DEF!   82H              accuracy ;LSB-NS!-NSB-
                                EXP
        DEFW   1
DECOR                           ;Result field
ATED                            ; after conversion contains
                                the value 2
```

Integer in Number of Easy Precision

CALL **0AB1H**

The content of workspace 1 is converted from a 6ance <Integer) to a simple precision fly8.

Example: The number 18569 is voa Forat of the integer unto a Fliejkoa number of simple precision and to be transferred to the ZIERT field.

```
LD      A, 8%           iLSB v01I 18569
LD      47921H),A       in workspace 1 i1'5J
LD      A4,48~          v01I 18569
LD      (7922H1,A       in workspace 1; type
LD      Ai2             flag = Integer
LD      (78AFH),A
```

```
        CALL    0AB1H       ; value unchange
        LD      HL,         ;Transfer result to field
        VALUE               ZWERT
        CALL 9CBH

VALUE                       Result field
        DEFS 4
```

ZWERT contains the value 00H-12H-I IH-BFH
(LSB-NSB-NSB-EXP).
This corresponds to the number 18569H in the form
of a simple precision floating-point number.

CALL 0ADBH — integer in number of double precision

The content of workspace 1 is converted from an integer to a flow of double precision.

Example: The number 657 shall be converted into a coefficient of double accuracy 111.

```
        LD      A, 91H      ;LSB of 657
        LD      (7921H) ,A  in Workspace 1
        LD      A2          l'ISB of 657
        LD      (7922HliA   ;In Workspace 1
        LD      A,2         Type flag = Integer
        LD      17AFFl,A
        CALL    0ADBH       in the value of double Ganauty
        LD      DE,VALUE    Result in result field
        LD      11...791DH  transmit
        LD      BC,B        ; (length = 8)
        LDIR

VALUE   DEFS 8              ;Result field
```

ZERT contains the value 657 after conversion as a double precision
flow ratio
(8# 80N-08} 8i4 00i-40i-24#-8Ai)

The following three routines convert a numeric ASCII string into one of the three data types.

When entering **the** register pair H point to the beginning of the string to be converted, the conversion is terminated when the first non-numeric character is reached.

All registers will be changed.

CALL 1ESAH — Handle ASCII string in integer u11M11

The ASCII string addressed with H is converted to an integer.

The result is passed in register pair DE.

Example: The ASCII string '16544' should be converted to an integer.

```
        LD      HL,VALUE    Initial value address
        CALL    1E5AH       iASCII string in
        LD      <CURRENT,E  integer ;Result in result
                N           field

AIERT   DEFM    '16544'     {Output
        DEFB    8           value ;End Id

DECOR   DEFW    8           ;Result field
ATED
```

ZIERT contains after the action the value 1544 as a 2-byte binary 6ance number (= 4AM),

CALL QECH

Convert ASCII string to binary value of any type

Converts the 1it ll. addressed ASCII-string in one of the three binary data types **and so on.**

If the value **of the** ASCII string is less than 32768 and the ASCII string does not contain a decimal point, an exponent value of 'E' or 'D', and a type designation of **'1' or '! ',** so the conversion into a 2-byte integer.

If the value is greater than 3277 or if the ASCII string contains a decimal point, the exponent specification 'E' or the type designation **'!',** it is converted to a simple precision flowcode.

For example, an exponent specification of 'D' or a type designation '#' is converted to **a** double-precision floating-point number.

Example	'12345'		conversion to integer conversion
	'40051'		to simple precision
	'12.3'	–	conversion to simple accuracy
	'12345!		UIMlander to simple accuracy
	'123451'	–	Conversion to double precision
	'123E111		lllAlmond in simple accuracy
	'		UM!Action in double precision
	'123D111		
	'		

The result **is** passed to i • Workspace 1. **11** Type flag displays the type **of** result

Example

```
        LD      11... AIERT     ;Address ASCII
        CALL    IIE6CH          string ;in binary value
                                and handle

AIERT   DEFN '24657'
        DEFB 1                  ;ASCII
                                string ;End
                                Id
```

The string '24657' **is** converted into an integer and transferred to workspace 1 (7921H-7922Hl. The type flag at 78AFH **is** set to 12H (= integer).

CALL 0E65H — ASCII string in double precision

This is a prelude to *the* above. Routine at IE6CH. This forces a conversion to a flow of double precision, regardless of *the* size and configuration of the ASCII string '

Example:

```
        LD      11.,        ;ASCII string adrl!Size
        AIIERT              In double. Transform accuracy
        CALL    IE65H

AIIER                       ;ASCII-Str i
T       DEFN    '24567'     ng ;End Id
        DEFR    0
```

The '24567' string **is** converted into a double-precision string.

Binary value in ASCH1-tring **uwndel**

The following three routines convert a numeric value from **the** binary format into an ASCII string w11.

CALL 0FAFH — Content of **ll.** in ASCII and elsewhere

A binary value in **i**H register pair is converted to an ASCII string and displayed at *the* cursor position on **the** screen.

This routine **is** used by *VOii* BASIC interpreter to represent the line of a programme line on **dea** Bildsehire '

Example

```
        LD      ll., 3839H  ;Load Binary Value
        CALL    llFAFH      in ASCII and other
```

On dt!ll screens *the* value 12345 (= 3839) is displayed. - 128

CALL 132FH — Convert integer to ASCII

An integer (Integer l) in the workspace is converted to an ASCIIString and placed in memory at *the* point addressed with it H, the registers B and C should be set to a value greater when the line is inserted, and a coma or dot insertion into ASCII string should be suppressed.

The result **will not be** marked with a **Mi** end flag

Example:

```
                LD      11.456      {value to convert
                LD      (7921H),HL  in Workspace 1
                LD      BC,686#     u. C > Set 6 ;Address result
                LD      H,          field Convert value to
                CALL    STRING      ASCII
                        1J2FH
STRING  DEFS    6                   ;Result field
```

After *the* conversion, the result field contains the entry 31H—JIH-J4H-35H-36H-81 · **11456.**

CALL **0FBEH** — Flow rate in ASCII-5tring

Transform a Fliellcma number of simple or double precision into an ASCII string. The number of tiles shall be provided in workspace 1. The generated ASCII-string **is** passed to the print buffer for a formatted number output at 7931H.

The ASCII string is completed **with 8llf**, II. points to the end of the generated ASCII string **(llltO). In the case of a** field overflow, the character '%' **is** entered in the byte before the pressure buffer (792FH),

In this conversion, a comfortable formatting of the ASCII string to generate can be requested. This **is** controlled by entries in **the** registers **A, B** and C.

The individual bits of the A-tab have the following formatting effects:

7 bit	= 0	- Do not format.
	"	- formatting according to the set bits,
bit *b*	=	– a coma is inserted every 3 digits to separate the thousand values.
bit 5	=	– leading spaces of the ASCII string are replaced by '*'.
Bit 4		– before the number, the sign '$' must be printed.
Bit 3	=	– A '+' as a sign must be represented.
Bit 2	=	– The sign is to be shown behind the number.
Bit 1		– not used
Bit 0	=	– ASCII representation with exponent output

B register		Number of characters to output left of decimal point,
C register	=	Number of characters to output to the right of the decimal point.

When entering 0FBDH instead of 0FBEH, the formatting is suppressed.

Example:

```
LD      HL, VALUE       ilitDilde an ASCII string to
        CALL    0E6CH   simple precision
        CALL 0FBDH      {and back to an ASCII string

AVALT   DEFM '1234,56'  { Output
        DEFB 0          value ;End Id
```

The output ASCII string '12.5' is converted to a number of simple precision.
With dellt CALL OF RDH it follows the conversion into the original ASCII string, which is now in the range 7930H ..... , with a leading space and a closing 00H, Register HL contains the entry 7930H and Register DE the entry 7938H.
The resulting field has the following hexadecimal content: 20H-31H-32H-33H-34H-2EH-35H-36H-00H

Artificial routines

These routines perform arithmetic operations between *two* operands of the same data type.

The routines expect the operands in the specified registers or workspaces, and the type flag should be set to the appropriate data type before the call.

The division routines use the division subprogramme'll in the KDlllllllr range (7880H-788DH); this IIII.ißwill be there intact.

Routes for **processing** 6-integer numbers

The following 5 routines perform arithmetic operations between two 1-bit 6 numbers, The two operands are to be provided in the HL and DE register pairs, with the exception of one exception, **the** contents of DE remain unchanged; the result **is** usually returned in register pair H.

CALL 0BD2H

Add two integers

Adds the contents of the register pair DE to the contents of the register pair HL. *The* Su11111e 11is used in Hl. .

However, if the sound exceeds 213 (excessive noise), both values are first converted into a fliekoa number of simple precision and the operation repeated. In this case, the result then stands as a value of simple accuracy in workspace 1 of the kill area. Type flag at 78AFH is assigned '4'.

Example: The integers stored in VALUE1 and VALUE2 shall be added.

```
        LD      LT,VALUE1   1. Load Value
        LD      DE,VALUE2   2. Load Value
        LD      A,2         Type Flag = Integer
        LD      (78AFH1,A
        CALL    IBD2H       Add Values
        LD      A, {7NFH)   ;Load Type Flaq
        CP      2           ;Result= Integer?
        JR      NZ,SP       ; no,=> united.    accuracy
INT                         Yes!

VALUE   DEFW    15

VALUE2  DEFW    40
```

CALL 0BC7H

Z111ei 6a subtract numbers

Subtracts the value in DE from the value in It., The difference **will be a register pair. ,**

If an underrun occurs, i.e. the subtraction of two values of unequal sign results in an ert } 21, then both values are converted to fliekoa numbers of simple accuracy before subtraction is repeated. The difference is passed as a value of simple accuracy in workspace 1. One such case is **aa** type flag, which **is** set from '2' to '4'.

Exampl e From VALUEl, VALUE2 should be subtracted to the

```
        LD      11...VALUEl     Load Deleted
        LD      DE,VALUE2
        LD      A2              iType::Integer
        LD      (78AFH),A
        CALL    IBC7H           subtract illE from HL
        LD      A, (78AF%)      ;Load type flag
        CP      2               Result = Integer?
        JR      NZ,SP           inein ==> simple precision
INT                             Yes!
VALUE1  IIEFW   50
VALUE2  IIEFW   30
```

CALL 0BF2H

l'lultiplikation of 2 integers

The content of II. 1it dell content is copied from DE. The product is then in HL.

In case of overflow {product 213, both values are unaltered in flow coats of simple accuracy and the l'ultiplication is performed again. In this case the product is in workspace 1, the type flag contains the value *It*.

Example: The content of VALUE! shall be multiplied by VALUE2.

```
        LD      11.,VALUEl      Load Deleted
        LD      DE,VALUE2
        LD      A,2             ;Type flag = Integer
        LD      (78AFM),A
        CALL    8BF2H           Copy Values
        LD      A, {7BWFM)      ;Load type flag
        CP      2               ;Result= Integer?
        JR      NZ,SP           ;no=>. accuracy
INT                             5yes!
VALUE1  DEFW    18
VALUE2  DEFW    12
```

CALL! 24920H — Integer Division

The content of DE is divided by Hl,

Both values are converted to single-precision floating-point numbers before the division, and the quotient is also passed with simple precision in workspace 1. The TgFlag at 78AFH is assigned the entry '4'.

The content of DE and Hl will be destroyed.

Example VALUE1 shall be divided by VALUE2.

```
                LD      DE,(VALUE1)  ;Load Dividend
                LD      H,           ;Load Divisor
                CALL    2490%        ;Execute Division

VALUE1          DEAi    80
VALUE2          DEFW    4
```

CALL A39H — Comparison of two integers

The content of It. and DE are compared algebraically. Both register contents remain unchanged.

The result of the comparison will be handed over to i' A-Register and the status flags <Z = ZERO-flag, C = CARRV-Flag),

Hl > DE	A = 1 A	
It. = EN	= 8 A =	Z flag = 1
It < EN	-1	C flag = 1, S flag = 1

Example? The contents of VALUE! and VALUE2 shall be compared.

```
                LD      HI… (VALUE1)  Load Deleted
                LD      EN,
                CALL    0A439H        Compare Values
                JP      Z,EQUAL       ;=> VALUE1 =
                JP      C,SMALL       i=> VALUE1 <
LARGE                                 ;=> VALUE1 >
VALUE           DEFW
VALUE2          DEFW
```

Only i simple precision synthetic operations

Five other routines are available to combine simple precision floating-point numbers arithmetically.

These routines expect an Argu111ent in the RC and DE register pairs and the second argument in workspace 1 of the communication area. The result is provided in principle in workspace 1.

CALL 716H

addition of simple precision

Adding two simple precision floating-point numbers,

A sand shall be provided in BC/DE, and the second volume in workspace 1. Match *of* addition **the** Su111me stands **in** workspace 1.

Example:

```
                LD      HL, VALUE1    i1. Adr. value
                CALL    9B1H          Transfer to Workspace 1
                LD      H,VALUE2      i 2. value adr-essi
                CALL    9C2H          to BC/DE
                CALL    716H          ;Add both values

lrlERTl         DEFS    4               Deletes simple precision
 VALUE2         DEFS    4               Deletes simple precision
```

The result *of the* addition is in the work area 1.

CALL **713H** simple precision subtraction

Subtracts a Flidko111111number of simple precision in BC/DE from the content of workspace 1, the difference is then **in** workspace 1.

Example'

```
                LD                  1, Address value
                LT,VALUE1           Transfer to workspace 1 2.
                CALL   9BIH         Address Value
                LD     HL,          in BC/DE carry iber
                VALUE2              ;VALUE! - VALUE2
                CALL   9C2H
                CALL   716H
VALU                                Provides simple accuracy
E!                                  for easier accuracy
VALU            DEFS   4
E2              DEFS 4
```

simple-precision multiplication

CALL! O847H

litiltiplicates the ia workspace 1 value of simple accuracy so **your** content from BCIDE. The protein is then placed **in** workspace 1.

Example:

```
                LD     H...VALUE1   1 . Address Value
                CALL   9BIH         Transfer to workspace 1 2.
                LD     LT, VALUE2   Address Value
                CALL   9C2H         ;to BC/DE
                CALL   847H         ; VALUE1 t VALUE2

VALU            DEFS 4              Illustrates          simple
E1              DEFS   4            accuracy   value     simple
VALU                                accuracy
E2
```

CALL 2490H Simple precision division

Divides a flow rate in BC/IIE0111111number of simple precision by the content of workspace 1 (simple precision), The ratio is then in workspace 1.

Example:

```
                LD              ;Address divisor
                1.,VALUE1       to workspace 1 ;address
                CALL 9B1H       dividend
                LD              i in BC/IIE
                11.,VALUE2      ;VALUE2 / VALUE1
                CALL 9C2H
                CALL 2490H

                                ;simple  accuracy  ;value
VALUE
1               IJEFS  4        simple precision
VALUE           IJEFS 4
2                               Comparison of values of simple accuracy
```

CALL ACH

This routine performs an algebraic comparison of two flow-coma numbers, these **must** be provided in the BC/DE and in workspace 1

The result of the comparison will be handed over in the A-Register and in the Flag-Register.

```
BC/EN  >  Arb. above.1      = - 1      Carry + sign flags = 1
BC/EN  <  Arb.ber.1         A = 1
BC/EN  =  Arb.ber.1         A= @       Zero flag = 1
```

Example

```
                LD      11...VALUEl     ;1. value to be compared
                CALL    9B1H            in Workspace 1
                LD      ll.,VALUE2      2. value to be compared
                CALL    9C2H            ;in BC/EN
                CALL    0CH             ;Compare values
                JP      7               ;=> VALUE1 = VALUE2
                JP      C...            i=> VALUE1 < VALUE2
                                        ;=> VALUE1 > VALUE2
VALUE1          DEFS    4               ;value of simple precision
VALUE2          DEFS    4               Simple precision
```

As for the two previous data types, five routines are available for the two double-precision floating-point measures to be combined.

An argument shall be provided in working area 1 1791DH-7924H> and the second argument **in** working area 2 17927H-792EH). The result is available **within the** working area 1.

CALL 0C77H

addition of double precision

Here **who** adds **the two** double-precision floating-point numbers and passes **the sum in** workspace 1.

Example

```
        LD      A,8          ;Type flaq = Double precision
        LD      178AFH1,A    ;Set
        LD      DE, VALUE1   t. Address argument
        LD      HL, 791DH    ;Address workspace 1
        CALL    9D3H         ;1. Argument in Workspace 1
        LD      DE,VALUE2    2. Address Argument
        LD      HL.7927H     ;Address workspace 2
        CALL    9DJH         ;2. Argument in Workspace 2
        CALL    8C77H        3VALUE] + ERT2

VALUE!  DEFS    8            doubles accuracy and
VALUE   DEFS 8               doubles accuracy
2
```

CALL C7H

double-precision subtraction

Subtracts an i1 workspace 2 Fliell!OMazal of double accuracy **of** content of workspace 1. The difference then is ia workspace 1.

Example:

```
                LD      A,8          ;Type flag = Double precision
                LD      (78AFH),A    ;Set
                LD      DE,          1. Address Value
                LD      HL,          ;Address workspace 1
                CALL    9D3H         1. Value in Workspace 1
                LD      DE,VALUE2    ;2. Address Value
                LD      HL.7927H     ;Address workspace 2
                CALL    9D3H         2. Value in Workspace 2
                CALL    0C70H        ;VALUE! - VALUE2

VALUE1          1IEFS   8            t double accuracy

VALUE2          DEFS    8            ;Double Precision Value
```

CALL 0DA1H double-precision zero replication

Multiplies the double-precision floating-point number in workspace 2 with dell content in workspace 1. The product is then i1 workspace 1.

Example:

```
                LD A8
                LD      (78AFH),A    Type Flag = Double Precision
                LD      DE, VALUE1   3
                LD      11th, 791DH  1. Address value ;Address
                CALL 9D3H            workspace 1
                LD      DE,VALUE2    1. Value in Workspace 1
                LD      11.7927H     2. Address value ;Address
                CALL 9D3H            workspace 2. Value in
                CALL BDA1H           workspace 2 ;VALUE1 t VALUE2

VALUE           DEFS    8            Value of double precision
1               DEFS    8            Value of double precision
VALUE
2
```

CALL 0DE5H double-precision division

Divides a double-precision floating-point number by one in workspace 2. The ratio is then in workspace 1.

Example:

```
                LD      A,8             Type Flaq = Double Precision
                LD      (78AFH>,A       ;Set
                LD      DE,             ;Address dividend
                LD      H.,791DH        ;Address workspace 1
                CALL    9D3H            ;Dividend in workspace 1
                LD      DE,VALUE2       ;Address divisor
                LD      LT.7927H        ;Address workspace 2
                CALL    9D03H           ;Divisor in workspace 2

                CALL    IDE5H           ;VALUE1 / VALUE2

VALUE1          DEFS    8               doubled accuracy

VALUE2          DEFS    8               doubled accuracy
```

CALL **0A4FH** Double Precision Comparison

For example, if you compare Fliejkoa numbers of double precision, it will compare one another. These shall be provided in work areas 1 and 2. The result of the comparison is presented in the A register and in the flag register.

```
Arb.Ber.1 > Arb.Ber.2          = 1 A=
Arb.Ber.  1  =  Arb,           @A= -f      Zero flag = 1
Ber.2 Arb. Ber.1 {Arb.                     Carry + Sign Flag = 1
Ber.2
```

Example

```
LD      MA8             Type-Fla = Double Precision
LD      (78AFH>,A       1. Address value ;Address
LD      DE,IERT1        workspace 1 ;Address value 1
LD H, 791DH             in workspace 1
CALL 9D3                2. Address value ;Workspace 2
LD      DE,IERT2        Address value 2 in workspace 2
LD LT, 7927H            Compare value fit value 2
CALL 9D3
CALL    IIMFH
```

```
                JP      z           :=> VALUE! = VALUE2
                JP      c...        :=> VALUE1 < VALUE2
                                    ;=> VALUE! > VALUE2
VALUE1  DEFS    8                   doubled accuracy
VALUE2  DEFS    8                   doubled accuracy
```

Nathl!Aoutilll!ll

The following routines are used to calculate uthellatic functions. These get a call at111, except for one exception, only one argument to pass in workspace 1 **of the** communication area. The type **of the** argument should be specified i111 type flag at 78AFH.

CALL **0977H** absolute value of er111ittel **ABSIN)**

The i • workspace 1 value **is** traded **to** its positive equivalent page111. The result will then also be i111 workspace 1.

If in workspace 1 the negative integer -3278, then **the** result **is** passed as a flow-coin of simple accuracy. The type flag at 78AFH **is** corrected accordingly.

All data types are allowed as arguments.

Example: The absolute value *of the* simple precision flowrate coefficient in *the* ERTI **shall be** determined.

```
LD      A           ;Type flag = simple
LD      !78AFHl,A   precision ;set
LD      HL, VALUE1  {Address Argument
CALL!9RIH           ; and in workspace 1. ;Create
CALL!  8977         Absolute
```

```
VALUE   DEFR 0F2M          Simple precision value;= -
        DEFB 81H           94.3456
        DEFB 0BCH
        DEFB 87H
```

When *the* operation is completed, the value of 94,3456 in workspace 1 is in simple precision.

CALL 0B37H

Determine the nearest integer INT (N)

This routine determines the integer part of a floating-point number. This must be provided in workspace 1, the type flag must display the correct data type.

If the value size allows (-32768 to +3277), the result **will be** returned in the data type '6 number', otherwise the data type will remain unchanged. The type of the result can be determined in the type flag at 78AFH.

The result is in Working Area 1.

Rice game: Of *the* Flie.llk011111111number *of* simple precision in value 1, the integer should be determined.

```
        LD      , 4           ;Type flag = simple accuracy
        LD      (78AFH),A     set
        LD      HL, WERTI     {Address Argument
        CALL    91H           Transfer to Workspace 1
        CALL    IBJ7H         ;Determine integer
```

```
VALUE   · DEFB   01H          kert single precision =
          DEFB 6FH            12,589
            DEFB 49
        DEFB    84H
```

The result, the number 12, is the data type Integer in workspace 1, the TgpFlag at 78AFH has the entry '2'.

CALL 15BDH — Arcus tangent (ATN CN)

A tangent value of 111stored in workspace 1 as a floating-point number is *the* corresponding angle **ia** Bogenmal erai. The result is provided as a flow coefficient in the work area,

Example — From the tangent value stored in the TAN field Determine *the* angle in radians and transfer it to the field 'RAD'.

```
        LD A              ;Type flag = simple
        LD    (78AFH),A   precision ;set
        LD Il, TAN        ; T angens-llert
        CALL  09B1H       ;Transfer to workspace
        CALL  15BDH       1;Determine angle
        LD    Hl, RAD     ;Addressing Result
        LD    DE,         Field ;Addressing Workspace
        7921H             1 ;Result in Result Field
        CALL  9D3H

TAN                       tangent of 30 (0.57735)
        DEFB  3AH
        DEFB  ICDH
        DEFB  13H
        DEFB  8MH
WH                        ;Result field
EE      DEFS  4
L
```

After lodging the **above** Routine contains the field **'RAD' the** value of the angle **3 i** arc (91H-04/-@/-OH = **8,523598)**

1541H CALL — cosine of a corner of the COS (Nl

Determines the cosine of an angle indicated in arctic wet.

The angle is to be provided as a flow **number in** workspace 1, the result is also passed as a flow number in workspace 1.

Example: *The* cosine of the angle specified in field 'RAD' shall be determined and transferred to field 'KOS';

```
        LD      A, 4            ;Type flaq = simple accuracy
        LD      (784FH),A4      set
        LD      HL, WHEEL       ;Angle in workspace l
        CALL    9BIH            carry over
        CALL    1541H           ;Find Cosine
        LD      HL, KOS         ;Address result field
        LD      DE,7921H        ;Address workspace l
        CALL    9D3H            Apply {Result

WHE     DEFB 91H                3g radian
EL      DEFB    0AH             dimension
        DEFB     OH             (0,523598)
        DEFR 8M

        DEFS 4
KOS                             ;Result field
```

After the calculation, the cosine of the angle of 3g is placed in the field 'KOS'
(D7H-B3H-5DH-80H = 0.66025)

CALL 1547H — Determine the sine of an angle SIN (N)

The sine of an angle is determined.

The angle shall be provided in radians as a flow rate in working area l. The result is also in I.

Example: Angle from delll ia field 'RAD' the sine shall be determined and passed in field 'SIN'.

```
LD A                ;Type flag = Set simple
LD     (78AFH),A    precision
LD     H,RA         ;Angle in workspace 1 ;
CALL 9BlH           transmit
CALL 1547H          ;acquire the sine
```

```
                LD      H... SIN        ;Addresse Result
                LD      DE,             Field ;Address Workspace
                7921H                   1 ;Transfer Result
                CALL    9D3H

WH                                      ;3@° radian (= 0.523598)
EEL             DEFB    91H
                DEFB    0AH
                DEFB    86H
                DEFB    88
                                        ;Result field
SIN             DEFS    4
```

The field 'SIN' shall contain, after the calculation, the sine of 390,

CALL 1439H

Determination of exponential function e' EXP {N)

it11means the values of base N (e = 2,71828);

The argument N is provided as a flow$kona number of simple accuracy in workspace 1. The result is also passed in simple accuracy i• workspace 1.

Example: 1.5788 to be determined.

```
                LD      A,4             ;Type flag = simple
                LD      (78AFHl,A       precision ;set
                LD      H... EXP        ;Address exponents
                CALL    9B1H            ;Transfer to workspace
                CALL    1439H           1 ;Calculate function ;Address
                LD      DE,             workspace 1 ;Address result
7921H                                   field Apply result
                LD      11... ERG
                CALL    9D3H

EXP             DEFB    BDBH            Exponent (1,5788)
                DEFB    OFH
                DEFB    49}
                DEFB    81H

ERG             DEFS    4               ;Result field
```

The initial values X and Yshall be provided as a FlieOkDM number of simple accuracy.

The base 'X' is to be transferred to the stack area, *the* exponent 'Y' is to be provided in workspace 1. The result is calculated as a single-precision floating-point number in workspace 1.

A special feature must be taken into account. Since one of the Argu11ente will be deployed to detl Stack, the routine cannot be called 1tit CALL, because the return address will be the last entry on the stack. The return address u$ rather be written to the stack before the argument and the routine called with a JP. The following example illustrates this procedure.

Example: 1 should be calculated.

```
        LD      HL, RET     ;Return Address
        PUSH    ed          to write to the stack
        LD      1.4         ;Type flag = simple accuracy
        LD      (78AFHJ,A   ;Set
        LD      H, BAS      Address base value
        CALL    9B1H        Move to workspace 1
        CALL    944/06      ;Working area run the stack
        LD      HL,EXP      Address exponents
        CALL    9B1H        Transfer to Workspace 1
        JP      13F2H       4 Determine
RET                         return address

BAS     DEFW    0           Base value = t
        DEFW    85H

EXP     DEFW    0           Exponent = 4
        DEFW    8YH
```

CALL 0809H

Natural Logarithus LOG {M)

Determines the natural Logarith111s of a FlidkOMazimuth of simple accuracy, The argument is to be provided in workspace 1, the result ·is also passed as a simple precision flowline in workspace 1.

Example' **The** natural Logarith111s of 5 shall be determined.

```
        LD      A,4          ;Type flag = simple accuracy
        LD      (7BAFH1,A    ;Set
        LD      1i.,ARG      {Address Argument
        CALL    9B1H         Transfer to Workspace 1
        CALL    0889H        {nat. Calculate logarithm
        LD      DE,7921H     ;Address workspace 1
        LD      1i.L06       Address result field
        CALL    9DJH         ;Take result

ARG     DEFW    1            {argument = 5
        DEFB    02H
        DEFI    83H

L06     DEFS    4            Result field
```

CALL 13E7H

Determine root of N SQR (NI)

Determines **the root** of a value stored in workspace 1. The result is also returned in workspace 1.

Example The root of 144 shall be determined

```
LD      A,A          ;Type flag = simple
LD      (78AFHJ,A    precision ;set
LD      HL,ARG       ;Address argument
CALL    9B1H         Transfer to workspace 1 ;Draw
CALL    1JE7H        Root
```

```
AR6     DEFV 9          $Argument = 144
        DEFB
        IMH
        IIEFB 88H
```

After calculation, the result is t= 12) in workspace 1.

CALL 14C9H

Random number of1 means RND (NI)

Creates a random number between I and 1 or between 1 and N, depending on the value 'N', which in workspace 1 passes who the **au$.**

The resulting random number is returned as a simple-precision flow-coefficient in working area 1 and the type flag is dialled accordingly 10.

The passed-over argument 'N' is the range of the random number. If N = @, a random number **between 8** and 1 **is** generated. If N > 8, **a random number between 1** and N **er1 is passed** and **as a whole** number.

Example: A random number **between** 1 and 28 shall **be determined:**

```
LD      A2
LD      (78AFH),A       ;Type flag = Set 6 number N
LD      H,28
LD      (7921H),H       = Load 28
CALL    14C%            ;and in workspace
                        1;obtain random number
```

According to the calculation, a random number between 1 and 28 for simple accuracy in workspace 1,

The resulting random number is not a real random number, but **is** formed by a test algorithm from the last random number.

If you want to add your randomness, you can use a CAl ID:1f to set a new base value, which will be **the** current state of the Z80 Refresh register. The base value and the last random number generated are at 78AAH - 78ADH **ia** K01111t1infection range.

Yes The lower address range of the ZBI are in Ber steps so-called Restart addresses, which can be reached with a special 'RST' command.

The RESTART vectors H to 3Mi are derived to RA expansion outputs **at the** beginning **of the** communication range. From there, for the vectors BH - 28H, a jump in RON routines, which also fulfil useful functions for an 11machine language or Assenb] of the promoters,

It is also possible to use jump commands in the RA} extension outputs in your own routines. However, it should be noted that not all other routines listed in *the* chapters are easily usable, since they occasionally also make use of the Restart commands.

RST S

Check a character

This routine is used **by the** BASIC interpreter to check the syntax of an input line. It **will** pass through it. compared addressed characters **with** the characters following the RST B command. If both **are** equal, the RST **1** routine **is** automatically called and from there on the 2. Byte jumped back after **the** RST B command. The H register pair is on **dl!ll** next valid character obtained by RST 11, the A register contains this character.

If the two characters do not match, an error message "SYNTAX ERROR **is** generated and jumped back to **the** input phase **of the** BASIC interpreter.

Example

Verification by ll. text addressed, whether it consists of the characters 'A=B'.

```
LD      LT... TEXT    Address Text String
RST     B             Check iA
IIEFB   'A'
RST     B             ;= check
IIEFB   ....
RST     B             B Check
DEFM    'Ɔ'
```

If **one of the** characters does not match, a SYNTAX ERROR **is** created.

RST 10H — Find next valid character

This routine is used by BASIC interpreter to identify and accept the ninest valid character when analysing an InputZeelle.

The H register pair is *used* to address a text string. When 1111rd is inserted, the address in it is increased by 1. The next character addressed by HL will be loaded into the A register and the carry flag will be set according to *the* character type.

CARRV = @ -Non-numeric characters
CARRV = 1 - nu111 characters

The ZERO flag is set when the text end is reached, this is marked by a BYte = 00H.

When editing *the* text string, spaces and the control characters @9 {TAB) and 0 (LF) are not considered, they are simply ignored.

Example: H points to a programme line that assigns a value information. Row edited to equal sign. It 1st to check whether a variable or a constant follows.

```
NEXT    RST   18H          ;Load next character
        JR    NC,NONUN     non-numeric
        CALL  1E5AH        ;Load constant
        JP                 further
NOfUI   CP    '+'          is it a '+' sign?
        JR    Z,NEXT       ;yes, load next character
        CP    , ,          Is it a '-' / sign?
        JR    Z,NEXT       Yes, next character
        CALL  268DH        ;Variable in Variable Table
                           ;acquire
```

RST 1BH — DE ait HI. compare

This routine performs a logical comparison of the two register pairs DE and HI. through. It does not work correctly with sign **numbers**, for which the routine is to use **at 8439** {arithmetic comparison **H:DE)**.

The result **is** displayed in the ZERO and CARRV flags. The content **of the** A register **will be** changed.

CARRV = 1	II. < DE
CARRY =	IL >= DE
ZERO = 1	equals
ZERO =	

Example: It shall be verified that a DE value is in the range **200** -5080.

```
LD    H, 500      ;Load upper limit
RST   IBH         ait compare value in DE
JR    C, ERROR    value in DE > 5011
ID    HL.199      ;Load lower limit
RST   IBH         compare with value in DE
JR    NC, HOLDER  Value in DE 208
```

RST 20H — Create erai data type

The gpFlag at 7BAFH **is** evaluated and a numeric value is returned in A-Register depending on the data type displayed there. The flag register can also **be** evaluated.

Tu	Status	A-
integer	NZ, C, M, E	-1
string	3.C.PE	8
par.Accuracy.	MZ,P,0	1
dpp, enauigk.	NZ, NC, PE	5

The value passed to the i• A-Register corresponds to **the** type code in 78AFH - J,

The routine •**v011** BASIC interpreter is used to determine the data type of a value stored in workspace 1, but be careful, type flag and workspace 1 do not have to be in sync.

Example: After adding two integers, it is to be checked whether the result **was** passed as a integer or a floating-point number of simple precision.

```
        LD      A,2          ;Type Flag = Integer
        LD      (78AFHl,A    ;Set
        LD      DE, VALUE1   1. Load Susands
        LD      HL, VALUE2   2. Load Suan
        CALi.   BBD2H        ;Addition of two integers
        RST     2            ;Type of result
        JP      M, INT       ==>Result= Integer
SP                           i=> Result=. accuracy

INT

VALUE1  IIEFW   2            ;1. U.S..and as Integer
VALUE2  DEFW    2            2. Number of seconds
```

on trapugs - routines

This section describes some routines that transfer data from different types within the memory or between registers and memory.

CALL **09B4H** Flickr count of simple accuracy of BC/IIE in workspace 1

transfers a file of simple precision from the BC/DE register pairs to workspace 1 of the caching area.

The content of **lt.** •Will be destroyed, BC/DE will remain unchanged.

Warning: Type flag at 7BAFH is not updated.

Example' Add two simple precision in-memory flow numbers.

```
        LD      EN, ZAll.1   i"5B + exponent of 1. Load Number
        LD      BC, ZAll.1   iLSB and NSB of 1. Load Number
        CALL    {09B4}       moved to workspace 1.
        LD      EN, ZAll.2   ;'5B + Exponent of 2. Load Number
        LD      BC, 1AL2+2   ;LSB and NSB of 2, Load Number
        CALL    0716#        ;Add to Workspace 1

ZAll.   IIEFS 4              ;value of simple
1       IIEFS  4             accuracy ;value of simple
ZAlt.                        precision
2
```

ZAH..1 and ZAHl.2 1must contain the values in the sequence LSB-NSB-11SB-EXP.

CALL 09B1H

Transfer simple precision flow coefficient to workspace 1

A simple precision floating-point number in memory is transferred to the workspace 1. H u$ contain the initial address of the memory area.

The contents of HL/BC/IIE .are destroyed.

Example

```
        LD      HL, VALUE    Move value to memory in
        CALL 9111H           workspace 1

VALU    DEFS                 contains a
E                            coefficient ;simple
                             precision
```

CALL **09CBH**

Apply Flow DialNumber of Simple Precision of Workspace 1

A floating-point number in workspace I will be placed in the programme memory over110111. HL IIIII contain the initial address of the memory area;

The content of all registers will be changed,

Example:

```
                LD      HL,     ;Load the memory
                VALUE           address ;Transfer value
                CALL    9CBH    from work to memory

VALU
E               DEFS    4       ;contains the flow ratio of
                                simple accuracy according to the
                                CALL.
```

CALL **09C2H**

Flows$coa number of simple accuracy from the memory to the BC/DE registers

Loads a FlielkQN number of simple precision from one memory area to the BC and DE register pairs.

H foot contains the initial address of the memory area.
The contents of all registers will be changed.

Example: Two numbers of simple precision shall be added. The result shall be reported in JC/DE,

```
        LD      11.,VALUE1  1. Address summands
        CALL    9B1H        in Workspace 1
        LD      HL, VALUE2  2. Addressing Suman
        CALL    9C2H        ;in BC/EN
        CALL    716H        ;BC/DE to Workspace 1 add.
        LD      BC, (7923M) Sun Exponent and NSB
        LD      EN, (7921H) Load iNSB and SUaSB

VALUE   DEFS    4           Simple precision

IERT2   DEFS    4           ;value of simple precision
```

VALUE! and VALUE2 contain the summands in the form LSE-NSB+ISR-EXP,

CALL 09BFH Simple precision floating point number from workspace 1 in BCIDE

transfers a simple precision floating-point number from workspace 1 to the BC/DE register pairs;

Warning, it is not checked whether the workspace 1 really contains **a** simple precision flowrate, hes nuss} of calling Pr0gram si are created.

Example: Multiply two floating-point numbers of single precision. The result shall be provided in BC/DE.

```
        LD      HL, VALUE1   ;Address Multiplier
        CALL    9BIH         Transfer to Workspace 1
        LD      HL, VALUE2   ;Address multiplier
        CALL    9C2H         ;BCIDE
        CALL    847H         ;Run Multiplication
        CALL    9BFH         ;Transfer product to BC/DE

VALUE   DEFS    4            Multiplicant in one. Exactly.
1       DEFS 4               Multiplier in Setup. Exactly.
VALUE
2
```

The entries in VALUE! and VALUE2 **1must** be present in the LSB-NSB—11SB—EXP format.

CALL 09A4H Transfer workspace 1 to the stack

is a flow count of simple accuracy from **the** workspace to the stack. This **is** stored there in order LSB-NSB—11SB—EXP.

All register contents remain unchanged.

It **is** not checked whether there really is a Fliefkona number of simple accuracy in workspace 1.

This routine **is** required for Belspiel if you use the potentier routine at 1JF2H 1110llen. There is the :base as Fliellk!lllllllll number of simple accuracy to provide on **the** stack.

CALL 09D3H

Variable Transmission Routine

depends on the data type, in the length of the type flag (7AFH) a value from the address specified in DE to that specified in II. specified address.

Registers A, DE and H. who changes the

Be1 game: A double-precision variable **11it de9 Na11en** 'XV' should be reported in the variable table and the value of the variable should be **transferred** to the **pr gra**

```
        LD      11.,NAME    ;Address variable names
        CALL    260DH       ;Determine Variable Address
                            ;DE contains the variable address
        RST     211H        Is it double accuracy?
        JR      MC, OK      ;yes, all right
        JP      ERROR       ;no, error
OK      LD      H,DP        ;Address storage area
        CALL    9DJH        ;Transmit Variable

NAI1E   DEFN 'XV'           Name of
        DEFB 8 DEFS         variable ;End ID
DP      8                   ;Variable receive field
```

CALL 29CBH

transfer a string variable

A String variable **is** transferred from **the** String area or the profile table to an internal storage area.

If the string variable is inserted, H contains the address of *the* variable *in* the variable table and DE contains the initial address ia PrograM.

Fr **a** strinq variable has a entry in the table of variables
Forat

```
1 • .Byte          =       Str Length
  2.+3.Byte        =     String Address
```

Example: A string variable with the manen **'4$'** should be transferred to a programmatic field 'VAR'.

```
            LD      HL,NAI'IE       ;Address variable name
            CALL    260DH           ;in table of variables
            RST     20H             ;Is it a Strlng variable?
            JR      Z,OK            yes, all right
            JP      FEil.ER         ;no, error
OK          EX      DE,HL           ;Variable tab address.      in HL
            LD      DE, VAR         ;Address header field
            CALL    29C8H           ;Transmit Variable
NAI1E       DEF11   'A$'            variables-ane
            DEFB    0               ;End ID
VAR         DEFS    255             5Receive Field
```

IASIC - FunktiDll!II

BASIC functions differ from previous functions mainly in the intensive use of the KOM1.1nikationsbereich.

When applying the following routines, the K011111Unication Area is intact and **has** not **been** destroyed or otherwise used by the calling 11**machine programme.**

The routines are appropriate. especially for use in 11aschinenprogra111111 subroutines called from **a** BASIC programme,

CALL 1B2CH

Get row in the programme111m

This routine searches the Progra11111r table for a BASIC row with a given line number. The line splitter to be determined should be provided in the DE register pair.

All registers will be changed. The status flags can be used to show the success of the action. BC and HI. contain corresponding address information.

Status	Flags	Register contents
Row found	C/Z	BC = Start address *of the* line in *the* programme table
Row not found **Lenumer** to **gr0$**	NC/Z	HL/BC = Programme end address + 1
Row not found but larger rows Now **in the** programme.	NC/NZ	BC = Address of the line ait ***of the*** next highest line zero'r HL = address **of the** following line in **Programme,**

Example: The programme line with *the* line number 58 should be **Pr gramme** to be calibrated. If there is one, the value % in *the* -Register is to pass *the* value -1 if it does not exist.

```
        LD      EN, 508     Loading          a
        CALL    1B2CH       line111111       in
        LD      A, 1        Progran
        JR      NC, A1      ID not found ;row not
        XOR     A           present
A1                          iA=I found for
```

The register pairs **are listed in** and BC contain the required address information **91!111à0** o.a. Table.

CALL 26DH

Determine the address of a variable

With this routine, the variable table can be searched for a besti1111th variable. The name of the variable you are looking for must be provided in a programme internal field and ■ addressed with HL.

If the variable does not exist, a new entry is made in the VariableTable. Value field is set = @.

If bouncing back is done, the DE register pair contains the address of the first value entry, the Tyrilag at 78AFH indicates the type of variable found.

Simple variable or elements of a matrix can be shared with this routine er11i. For a matrix the index is like for single!IR BASIC-access de¶ add names+ usually. 'A(20) for the 2nd element of matrix 'A',

Example: The address of the variable 'AB' should be ■.

```
        LD      HL, NAME    Address variable name ;Find
        CALL 260DH          variable address ;Retain
        LD                  address
        (ADR1,DE

                            ;Variable Na ■ e
NE      DEFM    'AB'        End ID after the
        DEFR 0              call
ADR     DEFV    0           ;address of variable 'AB'
```

GOSUB - E ■ ulation

CALL L00

This routine allows to call a BASIC–\jnterroutine from a, **llaschinenprgr on,** March execution of the BASIC routine is continued the programme ■ it de № on the CALL following command.

All registers !lll!rden changed.

At 111U8 HI. contain the initial address of an ASCII string specifying the first row1111111er of the BASIC subroutine.

Example: A BASIC\Jnterroutine, starting with Line **Muller 800,** should be called from **a** machine programme

```
        LD      1-I, ROW        Address Line Numbers ;Call
        CALL    L00             BASIC Routine

ROW     DEFN    '800'           il, BASIC routine line urator
        DEFB    0
```

This is just a small snippet of the ***available*** routines of the LASER-ROM. By studying **the** documented RN collection intensively, a variety of other routines can be localised and used for a call from Naschinenprogra111111en.

The entry-level models for students

Importer General CE TEC Trading GmbH Lange Reihe 29 2000 Hainburg 1

active computers

The Horne computers Laser 110, 210, 310 and VZ 200 owe their popularity not least to the comfortable and extensive BASIC interpreter, which is located in memory modules (ROMs) inside the computer and the user in all its functions after switching on is fully available.

The aim of this book is to describe the essential functions of the BASIC-ROM, so that you can better understand the internal processes in your computer and make the most of all functions. The book is also intended to give the assembler/machine programme expert the opportunity to use the functions of the BASIC-ROM in their own programmes, if:

to perform simple data conversions, use the I/O interfaces, or not programme mathematical functions.

VOGEL-BUCHVERLAG
WÜRZBURG

ISBN 3-8023-0874-3